Chr. E. de Boer

MSX PROBEERBOEK



Educaboek

Chr. E. de Boer MSX - PROBEERBOEK

ISBN 90 11 008898

MSX - *PROBEERBOEK*

Chr. E. de Boer

Educaboek

EDUCATIONAL PUBLISHERS · EDUCAGROEP ·

Australia	AEP, Blackburn (Melbourne)
België	Plantyn, Deurne (Antwerpen)
Belgique	Plantyn, Bruxelles
BRD	Stam, Köln
France	Casteilla/Educalivre, Paris
Great Britain	Hulton, Amersham
	Stam Press, Amersham
	Stanley Thornes, Cheltenham
Nederland	Educaboek, Culemborg
	Educa Int., Culemborg
	De Ruiter, Gorinchem
Suisse	Delta & Spes, Denges (Lausanne)

Uitgave van Educaboek

ISBN 90 11 008898

Inhoudsopgave Bladzijde

Bladzijde

Woord vooraf

Leren programmeren is niet zo moeilijk als velen denken.
Dat geldt zeker voor het programmeren van een microcomputer in de programmeertaal BASIC, vooral dankzij de mogelijkheid het geleerde onmiddellijk in praktijk te brengen.

Dit boek buit deze mogelijkheid van de combinatie MSX-microcomputer*) BASIC uit en helpt de lezer zich de beginselen van het programmeren eigen te maken. Dat gebeurt door proberen aan de computer: gekozen is voor een leerweg, die voortdurend van de lezer verlangt dat hij de gelezen theorie toepast en aldus toetst aan de praktijk.

In dit boek wordt enkele malen verwezen naar het bij de MSX-computer behorende BASIC-boek, waarin alle BASIC-opdrachten en -functies beschreven staan. Het probeerboek vervangt dit handboek niet: veel details (en daardoor mogelijkheden) van de MSX komen in dit leerboek niet aan de orde. Het is van groot belang het BASIC-boekje bij dit leerboek te gebruiken. Goed de weg weten in dit "naslagwerk" is later erg gemakkelijk. Daarom zijn in dit studieboek overzichten e.d. uit het BASIC-boekje bij de machine niet overgenomen. Wij laten het graag aan de lezer over het moment te bepalen, waarop zij of hij zelfstandig verder wil gaan met ontdekkend leren met het handboek als basis.

MSX is een standaard voor microcomputers. Bij het schrijven van dit boek heeft de auteur gebruik gemaakt van MSX-microcomputers van verschillende leveranciers. Het boek is geschikt voor al deze machines.

Het kan echter niet uitgesloten worden dat er microcomputers onder de vlag van MSX op de markt komen, die niet of niet geheel aan de standaard voldoen.

Noch de auteur, noch de uitgever kunnen aansprakelijk gesteld worden voor verschillen tussen dit boek en de werkelijke mogelijkheden van een MSX-microcomputer.

De MSX-microcomputers bieden de mogelijkheid tot grafisch werken en tot het programmeren van muziek. Aan beide mogelijkheden wordt in dit boek enige aandacht besteed, maar deze onderwerpen worden niet uitputtend behandeld. Het boek zou te omvangrijk worden. De auteur is daarom uitgegaan van het standpunt: eerst de "gewone", toch al zeer uitgebreide BASIC van de MSX leren kennen om dan later te leren werken met MSX-grafiek en MSX-muziek.

Veel plezier als MSX-BASIC-programmeur.

Christiaan de Boer   Ugchelen, april 1985

*) MSX is een handelsmerk van Microsoft Corp.

INLEIDING

1. Voor wie met wat?

Dit boek is bedoeld voor iedereen die wil leren programmeren met een MSX microcomputer. We kunnen veel plezier en nut hebben van de MSX als we hem programma's laten uitvoeren, maar nog veel leuker wordt het als we zelf in staat zijn de computer te laten doen wat wij willen. Daarvoor moeten we hem kunnen programmeren. De MSX wordt geprogrammeerd in de programmeertaal BASIC.

We gaan er in dit boek van uit dat we de MSX bij de hand hebben en alles meteen kunnen uitproberen. Dat is de beste manier om de computer goed te leren kennen. We gaan er ook vanuit dat de microcomputer klaar staat voor gebruik. Het boekje dat bij de computer hoort, vertelt hoe dat gedaan wordt.

Zet de machine en het tv-toestel aan. Het "beginbeeld" is bij alle MSX-computers hetzelfde:

MSX systeem
version 1.0
Copyright 1983 by Microsoft

Dit "beginbeeld" blijft maar kort zichtbaar en wordt gevolgd door het startbeeld. Dit startbeeld is niet bij alle MSX-computers gelijk. Sommige machines komen na het beginbeeld direct met het startbeeld van BASIC:

MSX BASIC version 1.0
Copyright 1983 by Microsoft
28815 bytes free
Ok
█

color   auto   goto   list   run

Als dit BASIC-startbeeld niet automatisch verschijnt moeten we eerst voor BASIC kiezen. In het boekje dat bij de computer hoort lezen we hoe dat moet.

In het BASIC-probeerboek gaan we steeds uit van het BASIC-beginbeeld:

MSX BASIC version 1.0
Copyright 1983 by Microsoft
28815 bytes free
Ok
▮

color auto goto list run

De getallen op dit beginbeeld kunnen anders zijn dan hier aangegeven is:

a. het versienummer (hier 1.0) geeft de versie van de BASIC-interpreter in de machine aan. De BASIC-interpreter is een programma dat in de computer is ingebouwd. Dit programma interpreteert opdrachten in BASIC, zodat de microprocessor deze opdrachten kan uitvoeren.
   Dit BASIC-probeerboek is geschreven bij versie 1.0 van de BASIC-interpreter.
b. Het grote getal (hier 28815) geeft het aantal "vrije plaatsen" in het geheugen van de computer aan. Op elke vrije plaats kan een letter, een cijfer of een BASIC-opdracht staan. Een "grote" MSX-computer heeft meer geheugen dan een "kleine". Kijk op het scherm: daar staat de geheugengrootte van de MSX-computer. Hoe groter het geheugen, des te groter kan ons programma zijn.

Opmerking: "Microsoft" is een bekend Amerikaans "softwarehouse". Dit bedrijf is de maker van de BASIC-interpreter in de MSX-computer.

Aan dit beeld zien we dat we contact hebben met de BASIC-interpreter in de machine.

Als vierde regel staat op het scherm: **Ok**

Dat is een belangrijke mededeling: hij vertelt ons dat de computer (of eigenlijk de BASIC-interpreter) klaar staat om onze opdrachten in ontvangst te nemen.

De regel vertelt ons ook dat de MSX op dit moment niets te doen heeft: hij staat te wachten. We kunnen het ook zo zeggen: altijd als er als laatste bericht **Ok** op het scherm staat zijn wij aan de beurt om iets te doen. Op de onderste regel van het scherm staan vijf woorden:

color auto goto list run

Deze vijf woorden horen bij de vijf "functietoetsen" van de computer. Dat zijn de vijf toetsen f1, f2, f3, f4 en f5 (of: F1, F2 ... enz.). In hoofdstuk 1 is daarover meer te lezen.

2. Aan, uit, RESET

Een belangrijke (en gevaarlijke) toets of knop is de "RESET". Bij sommige MSX-computers zit hij op het toetsenbord en staat de naam "RESET" erop. Bij andere zit hij op een veiliger plaats, achter op de machine. Bij bijna alle machines is hij rood: pas op!
Nu drukken we op de RESET. Druk de toets flink in. Wat gebeurt er?

Het beeld van de tv verdwijnt
even en komt daarna weer terug





Wat is er nu door het indrukken van de RESET-toets gebeurd? Met de RESET-toets beginnen we altijd weer helemaal van voren af aan: alles wat er in de MSX aan opdrachten zit verdwijnt. Eerst komt het beginbeeld op het tv-scherm, daarna het startbeeld. Zorg dat het "BASIC-beeld" weer op het scherm verschijnt als dat niet automatisch gebeurt. Op de vierde regel staat dan weer:

Ok
█

Wij zijn weer aan de beurt.

Als we nu aan het werk gaan moeten we voorzichtig zijn met het indrukken van de toets RESET. Als we op RESET drukken is de MSX "leeg". Alles wat we er al ingezet hebben is verdwenen en we beginnen weer met niets.

We moeten ook voorzichtig zijn met het uitzetten van de MSX. Als we de MSX uitschakelen met de schakelaar of door de stekker uit het stopcontact te trekken is het geheugen ook leeg. We zijn dus het programma dat in de machine zat kwijt. Tussen haakjes: als ons programma ook op cassette staat blijft het daar gewoon op staan.

Het kan wel eens gebeuren dat de stroom uitvalt. Daar kunnen we maar één ding van zeggen: als ons dat overkomt hebben we pech gehad. Het effect is hetzelfde als bij het uitzetten van de machine of het gebruiken van de RESET-toets. Overigens, als we tijdens het werk met de MSX het tv-toestel uitzetten geeft dat geen problemen: de MSX werkt gewoon door.

Dit boek is zo geschreven dat we alles meteen kunnen uitproberen met de MSX met kleurentelevisietoestel. We zullen daarom ook aandacht besteden aan bijvoorbeeld het maken van kleurenbeelden met de computer. We raden aan om vooral in het begin precies te doen wat er in het boek staat. We hoeven niet bang te zijn om fouten te maken: door fouten kan de MSX niet kapot gaan. We zullen af en toe zelfs met opzet fouten maken om te zien hoe de MSX daarop reageert.

Als we die reacties kennen kunnen we fouten die we per ongeluk gemaakt hebben gemakkelijker herkennen. Veel plezier als MSX-BASIC-programmeur!

3. De cursortoetsen

De MSX heeft een uitgebreid toetsenbord. Hieronder staan voorbeelden:





De toetsen op het toetsenbord hebben verschillende betekenissen, die soms op de toets staan aangegeven en soms niet. Alle "gewone" toetsen (de toetsen met een letter, een cijfer of een leesteken) krijgen een andere betekenis als ze gebruikt worden samen met de toetsen:

SHIFT (hoofdlettertoets)
CAPS (ook een hoofdlettertoets)
GRAPH (grafische toetsen)
CODE (speciale toetsen)

In deze paragraaf gaan we de toetsen en de combinaties met de SHIFT, CAP of CAPS, GRAPH en CODE uitproberen.

Doe precies wat in dit boek staat. Als op het scherm iets anders komt dan hier voorspeld wordt is er (voorlopig) maar één oplossing:

druk op RESET
ga naar het beginbeeld van BASIC
en probeer het opnieuw

Het is belangrijk deze paragraaf zorgvuldig door te werken: MSX heeft zoveel mogelijkheden op het toetsenbord, dat we gemakkelijk in de war raken. Neem er dus even goed de tijd voor.

Op het scherm staat:

MSX: **Ok**
█

Het witte blokje op het scherm is de "cursor", de "aanwijzer". Dit blokje geeft aan waar we kunnen gaan typen.

Druk op de letter k (op de toets staat een hoofdletter, maar als we deze toets indrukken komt er een kleine k op het scherm). Houd de k ingedrukt en kijk intussen naar het scherm.

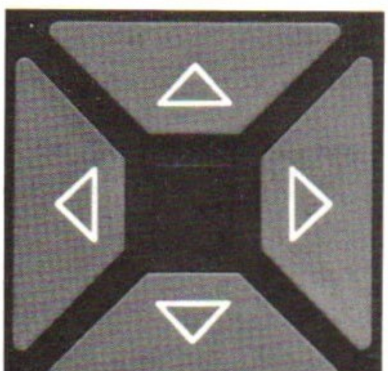
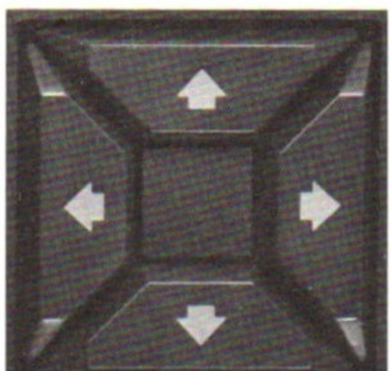
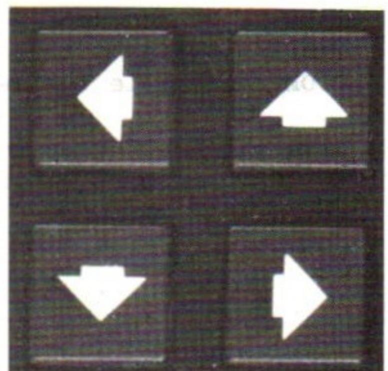
Wat gebeurt er? De MSX zet allemaal k's op het scherm en laat een tik horen bij elke letter:

MSX: **kkkkkkkkkkkkkkkkkkkk** enzovoorts.

Houd de k ingedrukt tot het hele scherm ermee vol staat. Alleen de laatste regel (met color auto enz.) blijft steeds staan.

Op één regel staan 37 letters k naast elkaar. Onder elkaar staan 23 regels.

Het scherm staat nu vol of bijna vol. Dat is een mooie gelegenheid om een paar andere toetsen uit te proberen. We beginnen met de cursortoetsen. Dat zijn de vier toetsen met pijlen.

Druk driemaal op de toets met de pijl naar boven:

MSX: het witte blokje gaat drie stappen (regels) naar boven.

De cursortoetsen verplaatsen de cursor. Druk nu driemaal op de letter p:

Wij: **ppp**
MSX: **pppk**

Drie letters k worden vervangen door drie letters p.

Druk nu drie keer op de cursortoets met de pijl naar links:

Wij: ← ← ←
MSX: **ppp**

De cursor gaat terug naar de eerste p. Typ nu:

Wij: **qqq**
MSX: **qqqk**

Druk nu drie keer op de cursortoets met de pijl naar rechts:

Wij:
MSX: **qqqkkkk**

Met de cursortoets laten we de cursor lopen. Zoek nu met de cursortoetsen uit:

a. Wat doet de cursor als hij helemaal linksboven op het scherm staat en we hem nog verder naar links of nog verder naar boven willen sturen?
b. Wat doet de cursor aan het eind van een regel als we hem nog verder naar rechts willen sturen? En op de laatste regel?
c. Wat doet de cursor als hij aan het begin van een regel staat en we hem nog verder naar links willen laten lopen?

Het antwoord op deze vragen is samengevat: de cursor kan overal naar toe gestuurd worden, maar hij kan niet van het scherm af!

Dit doen de cursortoetsen:

Druk op	De cursor
→	gaat één plaats naar rechts
←	gaat één plaats naar links
↑	gaat één plaats naar boven
↓	gaat één plaats naar onder

Opmerking: de cursortoetsen werken altijd hetzelfde, ook als ze samen met SHIFT, CAP(S), GRAPH of CODE gebruikt worden.

Doe nu het volgende:

Wij: **RESET** (niet typen, maar op de RESET-toets of knop drukken)
ga naar het beginbeeld van BASIC
MSX: **Ok**
█

Het scherm is nu bijna leeg. Probeer de vier cursortoetsen. Werken ze ook op een (bijna) leeg scherm?
Kunnen we met de cursortoetsen ook bij de tekst "MSX BASIC" komen? Zet de cursor dan eens op de X van MSX en typ een letter q:

Wij: q
MSX: **MSq█BASIC**

We kunnen dus veranderen waar we maar willen!

4. SHIFT en CAP(S)

Druk op RESET en ga naar het BASIC-beginbeeld.

MSX: **Ok**
Wij: **kkkk** (niet meer!)
MSX: **kkkk█**

Als we op het beginbeeld letters typen krijgen we kleine letters. Hoofdletters maken we met de combinatie SHIFT-lettertoets. Dat gaat zo (proberen):

Wij: druk SHIFT in en houd hem ingedrukt
KKKK
MSX: **kkkkKKKK█**
Wij: laat SHIFT los
kkkk
MSX: **kkkkKKKKkkkk█**

Dus: kleine letters zonder SHIFT, hoofdletters met SHIFT. De SHIFT wordt ingedrukt en vastgehouden.

Probeer nu het volgende: sla alle toetsen met een letter, een cijfer of een leesteken aan zonder SHIFT. Kijk wat het resultaat is. Sla daarna alle toetsen met een letter, een cijfer of een leesteken aan met ingedrukt SHIFT. Kijk wat dat oplevert.

LET OP: de toetsen met woorden erop (zoals ESC, TAB en CAP) mogen nog niet geprobeerd worden.

Opmerking: tussen de toetsenborden van de verschillende MSX-computers zitten kleine verschillen. Soms ontbreekt bijvoorbeeld het pond-teken £.

De toets CAP of CAPS lijkt op de hoofdlettertoets SHIFT, maar er zijn twee verschillen:

a. CAP of CAPS is een "omschakeltoets": hij mag niet vastgehouden worden, maar wordt aangeslagen. Als we dit doen gaat in de toets of op een andere plaats een lampje branden:







b. Als CAP(S) "aan" staat (het lampje brandt) krijgen we hoofdletters zonder de SHIFT te gebruiken, maar de cijfers blijven cijfers en van de leestekens krijgen we ook steeds de onderste van de twee die op een toets staan.

Eerst proberen:

Wij: **CAP(S)** (de toets aanslaan)
MSX: het lampje gaat aan
Wij: **abc**
MSX: **ABC**
Wij: **123**
MSX: **123**
Wij: **,.=**
MSX: **,.=**

CAP(S) schakelen we weer uit door hem nog eens aan te slaan:

Wij: **CAP(S)**
MSX: lampje gaat uit
Wij: **abc**
MSX: **abc**

Opmerkingen:

1. Als het op het scherm te ingewikkeld wordt om nog goed te kunnen zien wat er gebeurt, kunnen we met RESET opnieuw beginnen of met de cursortoetsen een plaatsje zoeken waar nog ruimte is.
2. In de voorbeelden en opdrachten laten we de cursor in het vervolg weg.

Wat gebeurt er als we SHIFT en CAP(S) met elkaar combineren? We zoeken dat uit met een paar toetsen: de letter k, het cijfer 7 en het leesteken ;

Vul het volgende overzicht in:

Opdracht 1:

CAP(S)	SHIFT	TOETS	RESULTAAT	
uit	niet ingedrukt	K		a
		7		b
		;		c
uit	wel ingedrukt	K		d
		7		e
		;		f
aan	niet ingedrukt	K		g
		7		h
		;		i
aan	wel ingedrukt	K		j
		7		k
		;		l

Het juiste antwoord staat bij de oplossingen achter in dit boek bij hoofdstuk 1, opdracht 1.

Opmerking: de combinatie van de ingedrukte SHIFT-toets met een andere toets noemen we vanaf hier kortweg SHIFT-toets. Een sterretje is dus bijvoorbeeld SHIFT-8 (druk SHIFT in, houd hem vast, sla het cijfer 8 aan, laat SHIFT los).

5. CODE

Met de toets CODE kunnen we letters met accenten en tekens uit het Griekse alfabet op het scherm krijgen. De toets CODE werkt niet op alle MSX-computers hetzelfde. Soms is het een toets die vastgehouden moet worden, soms is het een omschakeltoets met een lampje. Dat is gemakkelijk uit te vinden:

Wij: zet CAP(S) uit
     laat SHIFT los
     druk op CODE
MSX: lampje CODE gaat aan ... of niet!

Als er een lampje aangaat weten we zeker dat CODE een omschakeltoets is. Nog een keer op CODE drukken zet het lampje weer uit. Als er géén lampje is weten we nog niet helemaal zeker of de CODE nu wel of niet omschakelt. Dat kunnen we het beste proberen met één toets, de letter m.

Wij: **RESET** (voor de zekerheid)
MSX: **Ok**
Wij: **m**
MSX: **m**
Wij: **CODE** (aanslaan en loslaten)
     m
MSX: **m** (dan is CODE géén omschakeltoets) of ...
     μ (dan is CODE wél een omschakeltoets)

Als CODE géén omschakeltoets is, krijgen we de Griekse m (μ) zo:

Wij: **CODE** (indrukken en vasthouden)
     m
MSX: μ

Opmerking: als CODE een omschakeltoets is mag hij niet vastgehouden worden. Als we dat toch doen gaat hij steeds aan en uit. Probeer het zelf: houd CODE ingedrukt en druk m in en houd hem vast. Het resultaat is afwisselend m en μ: mμmμmμmμmμm enz.

Probeer nu alle toetsen met ingeschakelde of vastgehouden CODE-toets. Alleen de letter-, cijfer- en leestekentoetsen natuurlijk; niet de toetsen met de woorden als ESC of SELECT.

Opmerkingen:
1. Niet alle toetsen doen iets samen met CODE.
2. Niet alle "vreemde letters" zijn even goed leesbaar op het scherm.

We gaan nu combinaties van SHIFT, CODE en CAP(S) uitproberen met het cijfer 8.

Vul het volgende overzicht in:

Opdracht 2:
sla als toets steeds de 8 aan

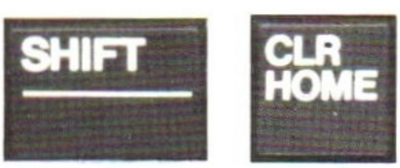
CODE	SHIFT	CAP(S)	RESULTAAT	
niet aan en niet ingedrukt	niet ingedrukt	uit		a
		aan		b
	wel ingedrukt	uit		c
		aan		d
wel aan of wel ingedrukt	niet ingedrukt	uit		e
		aan		f
	wel ingedrukt	uit		g
		aan		h

Met de toets n kunnen we maken: n, ñ en Ñ. Probeer dat uit met de toets n, SHIFT, CAP(S) en CODE.

6. Schoon schip: CLR

Met al onze probeersels komen er steeds meer tekens op het scherm, die we weer weghalen door de MSX-computer te "resetten" (op RESET te drukken). Dat hoeft niet op die manier, al heeft die het voordeel dat we zeker weten dat we weer met een schone lei in de "normale" beginstand beginnen. Schoonmaken kan ook met twee toetsen:

de terugtoets BS (van BackSpace) één plaatsje terug schoonmaken.
de wistoets HOME of CLR (HOME van terug-naar-huis, linksboven op het scherm, CLR van CLeaR, schoonmaken).





We gaan deze toetsen nu in de juiste volgorde proberen.

Wij: **HOME** (de toets)
MSX: de cursor gaat naar "huis", linksboven op het scherm

HOME brengt dus de cursor direct naar het begin van het scherm.

Wij: **SHIFT** (vasthouden)
HOME
MSX: het scherm wordt leeg (op de laatste regel na)

De combinatie SHIFT en HOME maakt het scherm leeg. Deze combinatie heet: wistoets (CLR). Het scherm is nu schoon. Om de BS-toets te kunnen proberen moet er wel weer iets op het scherm staan. Typ tien letters k (niet meer, niet minder).

Wij: **kkkkkkkkkk**
MSX: **kkkkkkkkkk**

Druk nu vijf maal op de toets BS (terugtoets)

Wij: vijf keer **BS**
MSX: **kkkkk**

Van de tien letters k zijn er nu nog maar vijf over. De toets BS wist ze, lopend van rechts naar links. We vullen nu de tien letters weer aan:

Wij: **ppppp**
MSX: **kkkkkppppp**

Laat de cursor nu met de pijl naar links teruglopen tot hij op de eerste letter p staat:

MSX: **kkkkkppppp** (de streep _ geeft de plaats van de cursor aan)

Druk nu één keer op BS. Wat verdwijnt er? Juist, een k. De toets BS wist naar links.

MSX: **kkkkppppp**

De cursor staat nog steeds onder de eerste p. Druk nu vier keer op BS:

MSX: **ppppp**

De cursor staat nu vooraan de regel. Druk nu nog eens op BS en kijk wat er gebeurt:

Wij: **BS**
MSX: **pppp**

De toets BS gaat nu verder met de tekens rechts van zich weg te vegen!

De toetsen SHIFT, CAP(S) en CODE hebben géén invloed op de werking van de toets BS. Bij elke combinatie zorgt BS voor het weghalen (wissen) van één teken links van de cursor. Pas als de cursor linksboven op het scherm staat begint BS naar rechts weg te halen.
Probeer nu:

Wij: **CLR** (wis het scherm met SHIFT en HOME)
MSX: (scherm leeg)
Wij: **het is vandaag valentijnsdag**
MSX: **het is vandaag valentijnsdag_**
(_ is de plaats van de cursor)

Laat de cursor nu naar links lopen tot hij onder de d van dag staat:

MSX: **het is vandaag valentijnsdag**

Haal nu met de toets BS valentijns van het scherm tot dit overblijft:

MSX: **het is vandaag dag**

Laat dat op het scherm staan!

De combinatie SHIFT-HOME (druk SHIFT in en houd hem vast, sla HOME aan, laat SHIFT los) noemen we vanaf hier **CLR** (van CLeaR). Deze afkorting staat niet bij alle MSX-computers op de toets.

7. INS en DEL

Bij de toets HOME zitten drie andere toetsen, die het typen op de MSX vergemakkelijken: INS van (INSert, invoegen), DEL (van DELete, weghalen) en STOP.





Met de toets INS kunnen we iets invoegen of tussenvoegen in een tekst op het scherm. Daar staat nog:

MSX: **het is vandaag dag** (_ is de plaats van de cursor)
Wij: **INS** (de toets)

Op het scherm krijgt de cursor een andere vorm: in plaats van een blokje d staat er een streepje onder de d van dag.
Dat is het signaal voor: PAS OP! We gaan invoegen!

Wij: **donder**
MSX: **het is vandaag donderdag**

De donder is vóór de dag ingevoegd. Laat nu de cursor terug lopen tot de d van donder:

MSX: **het is vandaag donderdag**

Kijk naar het scherm. Hoe ziet de cursor eruit?
De cursor is weer een blokje geworden. Dat betekent: we zijn niet meer aan het invoegen!

Wij: **zater**
MSX: **het is vandaag zaterrdag**

Zater is niet ingevoegd, maar over donde heen getypt.
Alleen de r van donder is blijven staan. Nu is zaterrdag wel een merkwaardig woord. Hoe maken we daar een fatsoenlijke zaterdag van?
Dat kan op twee manieren: met BS en met DEL.
We proberen het eerst met de bekende toets BS. Daarmee halen we een letter links van de cursor weg. Omdat de tweede r van zaterrdag weg moet, moet de cursor dus op de d van dag staan. Zet hem daar met de cursortoets neer.

MSX: het **is vandaag zaterrdag**
Wij: **BS**
MSX: **het is vandaag zaterdag**

Zet de r er nu weer tussen:

Wij: **INS** (de toets INSert)
MSX: **het is vandaag zaterdag**
Wij: **r**
MSX: **het is vandaag zaterrdag**

Zet nu de cursor met de cursortoetsen onder de tweede r van zaterrdag. De cursortoets wordt weer een blokje:

MSX: **het is vandaag zaterrdag**

Druk nu op de toets DEL van DELete (weghalen). Wat doet de computer?

MSX: **het is vandaag zaterdag**

De R is weggehaald ("ge-delete") met de toets DEL.
Druk nu nog drie keer op DEL. Dan verdwijnt de hele dag!

MSX: **het is vandaag zater_**

Als we nog een keer op DEL drukken gebeurt er niets meer!

Samengevat:

1. We typen een tekstje.
2. In dat tekstje kunnen we heen en weer lopen met de cursortoetsen.
3. Als we iets willen invoegen zetten we eerst de cursor op de plaats waar we moeten zijn. Dan drukken we op INS van INSert. We typen dan het woord op de woorden die we willen invoegen. Aan de cursor kunnen we zien dat we bezig zijn met invoegen: de cursor is een streepje geworden.
4. Iets weghalen kan op twee manieren: met de toets BS (van BackSpace) halen we tekens links van de cursor weg, met DEL halen we het teken waarop de cursor staat weg. Alles wat rechts van de cursor staat schuift op.

Probeer het zelf met de volgende zin:

Als het mooi weer is krijg ik altijd
zin om naar het strand te gaan.

Maak eerst het scherm schoon met CLR (SHIFT en HOME). Typ dan de zin.

Loop met de cursor naar de s van strand. Haal het woord strand weg met DEL. Voeg nu het woord zwembad in met INS. Kijk op het scherm; staat tussen het en zwembad een spatie? En tussen zwembad en te? Zorg ervoor dat ze er staan! Voeg nu tussen: het en mooi het woord heel in: eerst erheen met de cursortoetsen, dan INS van INSert, dan heel typen. Controleer de spaties!

Probeer nu van krijg ik te maken: krijgt mijn zus

Aanwijzing: ga met de cursor achter de g van krijg staan (krijg_), druk op INS, typ t mijn zus, druk driemaal op DEL. Klaar!

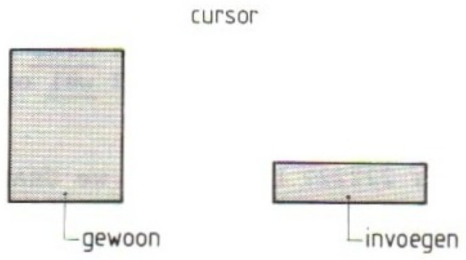
Ook nu kunnen we weer proberen of er bijzondere combinaties van toetsen zijn met INS, DEL en SHIFT, CAP(S) en CODE, Die moeite kunnen we ons besparen: INS en DEL blijven steeds hetelfde doen als we er SHIFT, CAP(S) of CODE bij gebruiken.

Er is nog wel iets anders waar we even naar moeten kijken. Als we op INS drukken gaan we invoegen. Dat is te zien aan de vorm van de cursor:

invoegen: de cursor is een streepje
niet invoegen: de cursor is een blokje

Met welke toetsen laten we de cursor weer veranderen van een streepje (invoegen) in een bokje (niet invoegen)?
Probeer het zelf:

Wij: **CLR**
MSX: scherm schoon
Wij: **INS**
MSX: cursor wordt streepje
Wij: één van de cursortoetsen
MSX: cursor wordt blokje
Wij: **INS**
MSX: cursor wordt streepje
Wij: **INS**
MSX: cursor wordt blokje
Wij: **DEL**
MSX: niets
Wij: **BS**
MSX: niets
Wij: **HOME**
MSX: cursor wordt blokje
Wij: **INS**
MSX: cursor wordt streepje
Wij: **CLR** (SHIFT en HOME)
MSX: cursor wordt blokje

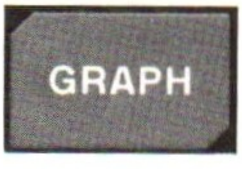


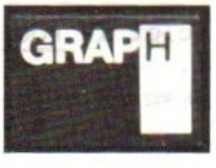
Conclusie: na INS wordt de cursor weer een gewoon blokje met INS zelf, HOME en CLR.

8. GRAPH

Met de toets GRAPH kunnen we grafische tekens (blokjes, lijntjes, gezichten) en een paar bijzondere tekens (bijvoorbeeld   en het wortelteken) op het scherm krijgen.

Net als bij de toets CODE werkt GRAPH op sommige MSX-computers als een omschakeltoets en op andere als de SHIFT-toets. Anders gezegd: bij sommige machines zetten we GRAPH aan en uit, bij andere moeten we hem vasthouden.







Hoe GRAPH werkt is gemakkelijk uit te vinden. Zorg eerst voor de normale beginstand: RESET de MSX-computer en ga naar het BASIC-beginscherm.

MSX: **Ok**
Wij: **GRAPH** indrukken en loslaten

Nu typen we het cijfer 2:

Wij: **22222**
MSX: **22222** (dan schakelt GRAPH niet om) **of...**
½½½½½ (dan schakelt GRAPH wel om)

Wat liet de MSX zien?

a. De cijfers 22222? Dan moeten we GRAPH vasthouden om de grafische tekens en dus ook de ½ op het scherm te krijgen.
b. De tekens ½½½½½ ? Dan is GRAPH een omschakeltoets: één keer indrukken zorgt voor grafische tekens (en dus ook de ½), nog eens indrukken zet GRAPH weer uit.

Probeer alle toetsen uit met ingeschakelde of vastgehouden GRAPH-toets. Probeer ze met en zonder SHIFT. Probeer ze ook met CAP(S) en CODE.

Opdracht 3:
Maak het scherm van de MSX schoon. Probeer daarna met GRAPH, de cursortoetsen en de toetsen met lijntjes een vierkant hokje midden op het scherm te tekenen.
Opmerking: de toetsen INS, DEL, HOME, CLR (SHIFT HOME) en BS blijven normaal werken als we ze samen met GRAPH gebruiken.

Opdracht 4:
Maak het scherm schoon en druk vervolgens drie of meer toetsen van het toetsenbord tegelijk in. Houd ze vast en kijk naar het scherm!
Probeer dat ook eens met twee cursortoetsen en met een letter- en een cursortoets.
Daarmee eindigt voorlopig onze verkenning van het MSX-toetsenbord. De overige toetsen komen later aan de beurt.

9. Samenvatting

We hebben nu kennis gemaakt met een aantal toetsen op het MSX-toetsenbord. We hebben nog niet alle toetsen gehad en toch is het al aardig ingewikkeld geworden.
Daar is niets aan te doen: het enige dat ons helpt is routine: als we vaak typen op de MSX gaan de problemen vanzelf over.
Het zal nog vaak voorkomen dat we in de war raken en bijvoorbeeld niet meer weten of GRAPH nu wel of niet aanstaat.

We kunnen maar één advies geven: probeer het steeds weer. Kijk naar het scherm. En denk na: wat heb **ik** gedaan dat mijn MSX zo reageert.

HOOFDSTUK 2: De MSX met teksten en getallen

10. Syntax-errors

Ga naar het beginbeeld van BASIC met RESET. De MSX staat klaar om voor ons aan het werk te gaan. Hij wacht op onze opdrachten. Dat kunnen we zien aan het woord **Ok** op het tv-scherm. Dit "Ok" betekent altijd dat wij aan de beurt zijn om iets te doen. Het witte blokje, de positiewijzer (Engels: cursor), geeft aan op welke plaats op het scherm de computer op ons staat te wachten. Begin maar eens met het typen van een naam:

Wij: **carmen**

Elke letter die we typen verschijnt op het tv-scherm. De positiewijzer schuift daarbij steeds één plaatsje op, zodat we kunnen zien waar we gebleven zijn. Nu staat **carmen** op het scherm met de positiewijzer daar direct achter. Klopt dat niet? Is er iets mis? Geen nood, we gaan gewoon verder.

De MSX heeft nog niet gereageerd op ons typwerk. Dat komt omdat hij nog niet weet dat we klaar zijn met typen. We hebben hem dat ook nog niet verteld en zolang we dat niet doen blijven wij aan de beurt.
We kunnen de MSX vertellen dat we klaar zijn met het typen van een regel als we RETURN gebruiken. "RETURN" staat erop.
Druk er nu op, dan zien we het volgende gebeuren:

1. De cursor springt naar het begin van de volgende regel. De MSX heeft onze boodschap ontvangen. Nu zijn wij even niet meer aan de beurt, maar de MSX is aan het werk.
2. Op het scherm verschijnt:

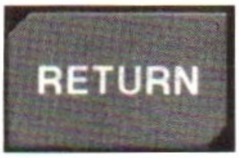
MSX: **Syntax error**
Ok

Wat is er aan de hand? Error is het Engelse woord voor fout. Er is dus iets mis. Dat komt zo: de MSX wil alleen opdrachten voor ons uitvoeren. En "carmen" is geen opdracht. Het is een prachtige naam maar de MSX begrijpt er niets van. Eigenlijk had hij dus moeten zeggen: IK BEGRIJP ER NIETS VAN of DEZE OPDRACHT KEN IK NIET of zoiets.

We mogen dus zeggen dat de regels Syntax error - Ok altijd betekenen dat er iets in onze opdracht zit, waardoor de MSX niet (precies) begrijpt wat we van hem willen. Hij doet dan maar liever niets en waarschuwt ons dat er iets fout is. Intussen staat er ook weer Ok: de beurt is dus aan ons.

Tot nu toe hebben we daar niets van gemerkt: de MSX heeft ons nog nooit verteld, dat hij ons niet begreep. En we hebben in de inleiding toch behoorlijk wat getypt.

Dat komt omdat we de computer nog nooit verteld hebben dat we klaar waren met typen en dat hij nu maar eens wat moest doen. Anders gezegd: we hebben de RETURN-toets nooit gebruikt!





Dat wordt nu anders! We gaan opdrachten typen en we sluiten elke opdracht af met de toets RETURN. Dat is voor MSX het sein dat we klaar zijn met het typen van een opdracht en dat hij aan het werk moet. Hij moet onze opdracht uitvoeren.
Als hij met zijn werk klaar is vertelt hij ons met Ok dat wij weer aan de beurt zijn.

Opmerking: in dit en de volgende hoofdstukken zijn de afbeeldingen van toetsen afkomstig van de Sony Hit Bit HB-55P en de Philips VG-8010. Op deze toetsen staan alle betekenissen van een toets. Raadpleeg zonodig het boekje bij de computer als u een toets niet kunt vinden.

11. De opdracht PRINT

De MSX kan alleen opdrachten voor ons uitvoeren. Als we hem iets willen laten doen moeten we dus altijd een opdracht geven. Opdrachten herkent de MSX aan bepaalde woorden. Die woorden moeten we gebruiken om hem voor ons te laten werken. Eén van die opdrachtwoorden is: PRINT

PRINT zegt de MSX dat hij iets op het tv-scherm moet laten zien. Dat zullen we hem laten doen.

Wij: **print "carmen"**

De aanhalingstekens moeten er wel bij staan, anders gaat het nog fout. De aanhalingstekens typen we zo:

1. Druk de hoofdlettertoets in en houd die vast.
2. Druk nu de toets met het dubbele aanhalingsteken " in. Tweemaal een enkel aanhalingsteken is niet goed.





Druk dan op RETURN. Daarmee zeggen we weer: Wij zijn klaar met typen, het is jouw beurt. Nu reageert de MSX heel anders. Kijk maar op het scherm.

Wij: **print "carmen"**
MSX: **carmen**
Ok

Klopt dat? Of staat er weer Syntax error? Dan hebben we het niet helemaal goed getypt en moeten we het nog maar eens proberen. Als het wel klopt heeft de MSX gedaan wat we vroegen. Hij staat weer klaar voor de volgende opdracht.

Laat de MSX nu een naam op het scherm zetten. Geef hem een opdracht met PRINT en een naam tussen aanhalingstekens en druk op RETURN. De MSX voert deze opdracht meteen uit en zegt dat hij klaar is voor de volgende. Lukt dat niet de eerste keer, probeer het dan nog eens.

Met een printopdracht kunnen we de MSX ook meer dan één woord op het tv-scherm laten zetten. Probeer:

Wij: **print "de eerste opdracht ken ik al"**
MSX: **de eerste opdracht ken ik al**
Ok

We kunnen een fout in een opdracht verbeteren zolang we de MSX nog niet met RETURN gezegd hebben dat we klaar zijn met het typen van de opdracht. Hebben we RETURN al gebruikt, dan voert de MSX de opdracht meteen uit, ook als we in de tekst een typfout hebben gemaakt. De MSX voert een opdracht niet uit als we in de opdracht zélf (hier in het woord PRINT) een fout maken. Hieronder staan een paar PRINT-opdrachten waarmee we fouten kunnen uitproberen. Andere teksten mogen ook! Schrijf op hoe de MSX reageert.

Wij: **print "dit is een"** MSX: ....

Wij: **plint "dit is twee"** MSX: ....

Wij: **print "dis twee"** MSX: ....

Wij: **print dit is drie** MSX: ....

Wij: **print "dit is drie"** MSX: ....

Wij: **print "dit is vier** MSX: ....

Wij: **print"dit is vijf** MSX: ....

Wij: **print "dit ijs is heet"** MSX: ....

Experiment klaar? Dan kunnen we een paar conclusies trekken:

1. Als we in het woord PRINT een fout maken begrijpt de MSX ons niet en reageert met "Syntax error".
2. Als we in de tekst tussen de aanhalingstekens een fout maken trekt de MSX zich daar niets van aan: hij voert de opdracht uit zoals we die gegeven hebben.
3. Als we het eerste aanhalingsteken weglaten begrijpt de MSX ons toch, we krijgen geen Syntax error. De tekst komt niet op het scherm: er staat een cijfer 0. Als we alleen het laatste aanhalingsteken weglaten gaat het wel goed.
4. We mogen tussen PRINT en de tekst die we willen laten zien een spatie zetten. Dat is voor ons gemakkelijker te lezen, maar voor de MSX maakt het niets uit: als we die spatie weglaten doet hij ook wat we vragen.
5. De MSX zet ook onzin op het scherm als we dat willen.

12. Strings

Voor we verder gaan moeten we een paar dingen op een rijtje zetten.

A. We zijn nu bezig de MSX direct voor ons te laten werken. Dat wil zeggen: de opdracht die we typen wordt onmiddellijk uitgevoerd. We zijn nog niet bezig met het maken van een programma.

B. De tekst tussen aanhalingstekens noemen we in BASIC een "string". Zo'n string staat altijd tussen aanhalingstekens. De aanhalingstekens geven het begin en het eind van de string aan, maar ze horen er zelf niet bij. We hebben dat ook kunnen zien: de aanhalingstekens komen niet op het scherm als we een PRINT-opdracht geven. In de string mag alles staan: We mogen er een naam van maken: CARMEN.
We kunnen er cijfers in zetten (zoals in een autonummer): FG-76-SG.
We kunnen ook "vreemde" tekens gebruiken:)(00%!?
In de string kunnen dus cijfers, letters, leestekens en speciale tekens staan. Er is eigenlijk maar één teken dat niet in een string kan staan: dat is het dubbele aanhalingsteken zélf. Dat gebruiken we om het begin en het eind van een string aan te geven.

Als we de aanhalingstekens in een string zelf zetten komt de MSX in de war. Probeer dat met de volgende PRINT-opdracht:

Wij: **print "in deze "string" zitten aanhalingstekens"**

MSX: ....

Let op! deze opdracht past niet helemaal op één regel op het scherm. Daarvan hoeven we ons niets aan te trekken. Typ gewoon door en druk pas op RETURN als de hele opdracht op het scherm staat!

Als we wat verder zijn kunnen we de reactie van de MSX wel verklaren. In een string mag wel een enkel aanhalingsteken voorkomen.

Wij: **print "in de 'string' zitten aanhalingstekens"**
MSX: **in de 'string' zitten aanhalingstekens**

13. Rekenen

Computers kunnen supersnel rekenen. De MSX kan dat ook. We zullen hem dat nu laten doen. Geef de MSX de volgende opdracht:

Wij: **5+7 RETURN**
MSX: **niets**





Geen reactie? Heeft de MSX de opdracht uitgevoerd? In ieder geval heeft hij ons begrepen, want er staat geen "Syntax error". Maar wat er precies gebeurt...
Nu willen we toch wel graag weten wat de uitkomst van dit sommetje is. Daarvoor moeten we niet alleen opdracht geven om de rekensom te maken, maar ook om de uitkomst van die rekensom te laten zien. Probeer dat zo:

Wij: **print 5+7 RETURN**
MSX: **12**
Ok

Dat ziet er beter uit! Als we in de opdracht geen typfout hebben gemaakt weten we nu dat vijf en zeven twaalf is en ook dat de MSX wel degelijk rekenen kan. Staat die uitkomst 12 er niet? Kijk dan nog eens goed naar de opdracht. Zit er een fout in? Laat het de MSX nog eens doen.

In de vorige PRINT-opdracht hebben we geen aanhalingstekens gebruikt. Vergelijk nu de volgende twee opdrachten. Laat ze allebei door de computer uitvoeren:

Wij: **print "5+9" RETURN**
MSX: **5+9**

Wij: **print 5+9 RETURN**
MSX: **14**

Bij de eerste opdracht zeggen we tegen de MSX: laat de string 5+9 op het tv-scherm zien, precies zo als we hem tussen de aanhalingstekens hebben gezet. In de tweede opdracht zeggen we: reken een sommetje uit en laat de uitkomst zien. In het eerste geval rekent de MSX niet, bij de tweede opdracht doet hij dat wel. Probeer nu de volgende opdrachten en schrijf de reactie van de MSX op.

LET OP: we zetten niet meer bij elke opdracht dat er op RETURN gedrukt moet worden, maar dat wil niet zeggen dat we het mogen vergeten! Wel is belangrijk te weten dat we een opdracht nog kunnen veranderen, zolang we niet op RETURN hebben gedrukt. Bij het veranderen gebruiken we de cursortoetsen, BS, INS en DEL.

Wij: **print 23+48** MSX: ....

Wij: **print 1+2+4+7** MSX: ....

Wij: **print 1+2 + 3 + 4 + 7** MSX: ....

Wij: **print "1+2+3"** MSX: ....

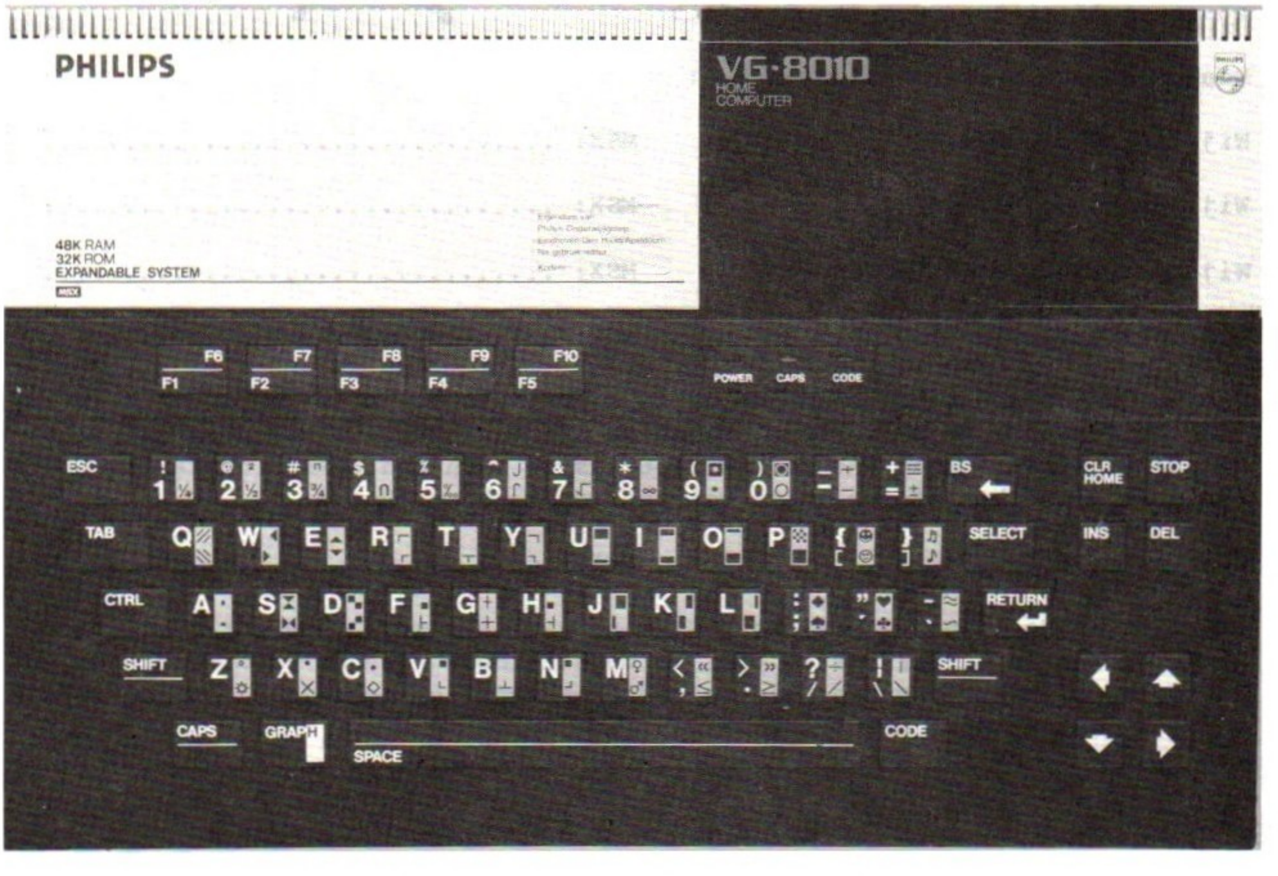
Wij: **print "1 + 2 + 3"** MSX: ....

Wij: **print1+2+3** MSX: ....

Wat is het effect van de opdrachten? Maakt het iets uit of de getallen wel of niet tussen aanhalingstekens staan? Maakt het iets uit of tussen het woord print en het cijfer 1 wel of niet een spatie staat?

Opmerking: om de 0 (nul) en de O (Oh!) goed te kunnen onderscheiden, zet de MSX een streepje door de nul:

O is Oh!
Ø is nul

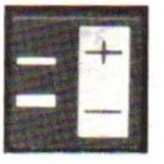


14. Aftrekken en de decimale punt

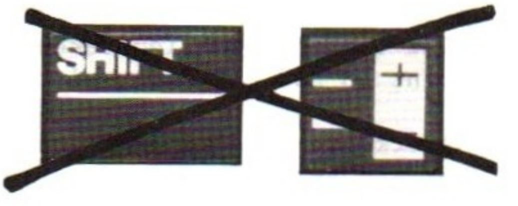
Laat de MSX nu aftreksommen maken:

Wij: **print 40-10** MSX: ....

Pas op met de min-toets. Er staan twee streepjes op: de onderste (zonder SHIFT) is een echte min, het bovenste (SHIFT erbij gebruiken) geeft een streepje onderaan de regel. Als we die per ongeluk gebruiken krijgen we dit:

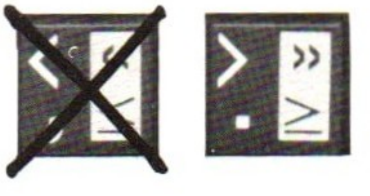


Wij: **print 40_10** (SHIFT-min gebruiken)
MSX: **40**
Syntax error
Ok



Uitleg van deze reactie volgt later!

Dat kan hij dus ook. We kunnen ook berekeningen maken met getallen met cijfers achter de komma. Alleen... de MSX gebruikt geen komma maar een punt. Het getal vijfentwintig-en-een-half ziet er dus zo uit: 25.5



Maak daar een paar rekensommen mee:

Wij: **print 25.5+24.5** MSX: ....

Wij: **print 25.5-24.5** MSX: ....

Wij: **print 82.731+0.03** MSX: ....

Wij: **print 82.731-.731** MSX: ....

Als we geen typfouten hebben gemaakt zien we, dat dit prima werkt. Als er alleen nullen vóór de punt staan, mogen we die weglaten. De MSX doet dat ook! Als we nu willen weten wat de MSX doet als we tóch een komma in plaats van een punt gebruiken moeten we de volgende opdracht proberen:

Wij: **print 8,2+7,3** MSX: ....

Dat ziet er heel anders uit! We zullen nu niet uitleggen hoe dat komt. Dat komt wel als we verder praten over de mogelijkheden van de PRINT-opdracht. Nu moeten we alleen onthouden: als we gebroken getallen gebruiken noteren we die met een punt en niet met een komma.

15. Vermenigvuldigen en delen

De MSX kan rekenen met heel grote getallen. Dat zullen we hem straks ook laten doen, maar in de volgende opdrachten staan nog wat kleine getallen. We kunnen dan gemakkelijk narekenen of de MSX zijn werk goed doet. Probeer:

Wij: **print 84+16** MSX: ....

Wij: **print 16+85-20** MSX: ....

Wij: **print 10000-5000** MSX: ....

Wij: **print 10.000-5.000+5000** MSX: ....

Denk eraan dat in de laatste opdracht de punt eigenlijk een komma is. Er staat dus 10 min 5 plus 5000 en niet 10000 min 5000 plus 5000.

In BASIC gebruiken we de + en de - zoals we dat gewend zijn. Voor vermenigvuldigingen gebruiken we een ander teken: het sterretje (de "asterisk"). Geef de volgende opdrachten:

Wij: **print 7*5** MSX: ....



Wij: **print 1.5*2** MSX: ....

Wij: **print 1000*.1** MSX: ....

De MSX kent de regel "Meneer Van Dalen Wacht Op Antwoord". Kijk maar naar de uitkomst van de volgende berekening:

Wij: **print 2+4*3** MSX: ....

De MSX berekent eerst 4 keer 3 is 12 en dan 12 plus 2 is 14. Straks komen we op Meneer van Dalen nog even terug. Nu is eerst de beurt aan delen. In deelopdrachten gebruiken we de schuine streep (zijn Engelse naam is slash). Probeer dat met de volgende opdrachten:

Wij: **print 4/2** MSX: ....

Wij: **print 7/5** MSX: ....



Wij: **print 10/4** MSX: ....

Wij: **print 2.7/.3** MSX: ....

Wij: **print 3/4-2/5** MSX: ....

Reken maar na: eerst 3 gedeeld door 4 is 0,75. Dan 2 gedeeld door 5 is 0,4. Daarna aftrekken 0,75 min 0,4 is 0,35. De MSX maakt ervan .35!

16. Haakjes

Als berekeningen wat ingewikkelder worden gebruiken we haakjes om geen problemen te krijgen met Meneer van Dalen.



Probeer de volgende twee opdrachten:

Wij: **print 18*100+5/100** MSX: ....

Wij: **print 18*(100+5)/100** MSX: ....

Het verschil is heel groot. Bij de eerste opdracht houdt de MSX zich aan Meneer van Dalen en berekent eerst 18 maal 100, daarna 5 gedeeld door 100 en telt dan de uitkomsten op. Bij de tweede opdracht berekent hij eerst 100 plus 5 en gaat daarna vermenigvuldigen en delen.

Bekijk de volgende opdrachten en ga na of ze verschillende uitkomsten opleveren. Daarna laten we de MSX het rekenwerk doen om te zien of we gelijk hebben:

Wij: **print 3*7+6*2** MSX: ....

Wij: **print 3*(7+6)*2** MSX: ....

Wij: **print (3*7+6)*2** MSX: ....

Wij: **print 3*(7+6*2)** MSX: ....

Wij: **print (3*7)+(6*2)** MSX: ....

De laatste opdracht is precies hetzelfde als de eerste. Als de MSX deze opdrachten heeft uitgevoerd zien we dat de uitkomst ook precies hetzelfde is. We mogen dus ook zeggen dat de haakjes in de laatste opdracht geen kwaad kunnen. Als we het zelf duidelijker vinden om de haakjes te gebruiken mag dat best, want de MSX heeft er geen last van.

Laat de MSX de volgende berekeningen maken:

Wij: **print 2/3** MSX: ....

Wij: **print 20/3** MSX: ....

Wij: **print 200/3** MSX: ....

De drie PRINT-opdrachten hebben een uitkomst van zes cijfers. Als we zelf deze deling uitrekenen krijgen we 20/3 = 6,6666666666666....

We gaan door met zessen achter de komma tot we vinden dat de uitkomst nauwkeurig genoeg is. Dan ronden we af.

De MSX vindt de uitkomst nauwkeurig genoeg als hij 15 cijfers heeft. Hij rondt dan af, zodat er 14 cijfers in de uitkomst overblijven.

Bekijk het afronden door de MSX met de volgende opdrachten:

Wij: **print 1.44444444444444** MSX: ....

Wij: **print 1.55555555555555** MSX: ....

WIJ: **print 1.000000** MSX: ....

De MSX doet het precies zoals wij gewend zijn: vijf en hoger worden afgerond naar boven, minder dan vijf naar beneden.
De laatste opdracht heeft laten zien dat de MSX de nullen achter de decimale punt weglaat.

Even een opmerking tussendoor. We hebben in de laatste bladzijden aan de mogelijkheid van syntax errors geen aandacht meer besteed, omdat we inmiddels weten, wat zo'n syntax error betekent: we hebben in de opdracht iets getypt dat de MSX niet begrijpt. We moeten de opdracht opnieuw (maar dan wél verbeterd) geven.

17. Machtsverheffen

We gaan de MSX nu wat zwaarder werk geven: we gaan machtsverheffen. Machtsverheffen is niets anders dan herhaald vermenigvuldigen, bijvoorbeeld:

2^{17}=2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2

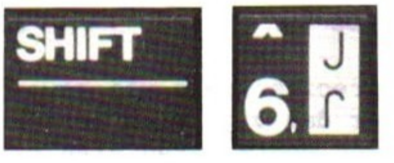
We kunnen de MSX deze berekeningen natuurlijk zo laten maken:

Wij: **print 2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2**
MSX: **131072**

LET OP: de som past niet helemaal op één regel. We trekken ons daar niets van aan en typen gewoon door. Druk pas op RETURN als de hele berekening getypt is.

Als we de opdrachten goed hebben gegeven zullen de uitkomsten ook goed zijn. Als wij gaan machtsverheffen moeten we herhaald vermenigvuldigen. De MSX doet dat ook, maar er is een gemakkelijker manier om hem te vertellen dat hij deze berekening moet maken. We hebben dan minder werk en vergissen ons minder gemakkelijk.

Voor machtsverheffen gebruiken we het teken dat op de toets met 6 zit (hoofdlettertoets indrukken en vasthouden, 6 aanslaan).



Wij: **print 3^2** MSX: ....

Wij: **print 5^3** MSX: ....

Wij: **print 2^17** MSX: ....

Geen syntax errors gekregen? Als dat is gebeurd hebben we de verkeerde toets gebruikt. Kijk nog eens goed waar die pijlpunt zit en probeer het opnieuw.

Opmerking: als de uitkomst van een berekening erg groot of erg klein wordt ziet hij er bijvoorbeeld zo uit: 9.22337E+18

Als we dit getal normaal zouden opschrijven zou het er zo uitzien
9.22337E+18=9223370000000000000

Zo'n merkwaardig getal met een letter E erin heet een "getal met een drijvende komma". Het eerste stukje van het getal heet de mantisse. In dit getal is dat 9.22337. Dan komt de letter E. Daarachter staat de exponent (hier +18). De waarde van het getal is **mantisse** maal 10 tot de macht van de **exponent.** In formule geschreven: $9.22337*10^{18}$

Denk er wel even aan dat de punt in de mantisse een **decimale** punt is.

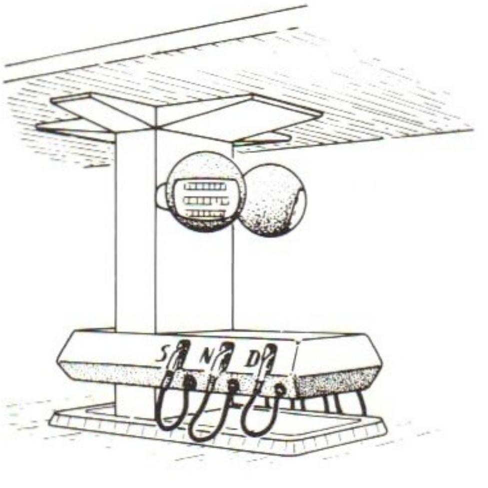
We sluiten dit hoofdstuk af met drie opdrachten om zelf uit te voeren.

Opdracht 1:
Jantjes fiets heeft wielen met een middellijn van 26 inch. Een inch is 2.54 cm. Hoe vaak draait Jantjes wiel rond, als hij de afstand van huis naar school (733 m) fietst? (Neem voor pi 3.142)

Opdracht 2:
Pa's auto loopt 1:12,2 als hij zich aan de maximumsnelheid van 100 km/uur houdt. Bij 130 km/uur gebruikt hij 1:8,9. Een liter benzine kost f 1,64. Pa rijdt per maand zo'n 3300 km. Hoeveel geld bespaart hij, als hij 100 rijdt i.p.v. 130?

Opdracht 3:
Ma doet aan de lijn en leest op een pak toastjes: "Een toastje bevat 45 kJ (11 kcal)". Hoeveel calorieën is één joule? En hoeveel joule is één calorie?

De oplossingen staan achter in dit boek.



18. Programmaregels

In het eerste hoofdstuk hebben we namen op het tv-scherm gezet met een PRINT-opdracht. We laten nu een programma zien, dat hetzelfde doet. Alleen: dit programma zet een naam op élke regel van het tv-scherm. Dit is het programma (nog niets typen, we bekijken eerst het voorbeeld):

```
10 PRINT "Carmen"
20 GOTO 10
```

Het programma bestaat uit twee opdrachten. De eerste opdracht zegt: zet de naam Carmen op het scherm:

```
10 PRINT "Carmen"
```

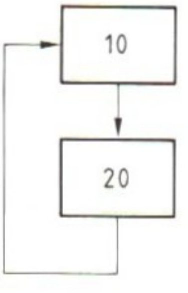
De tweede opdracht zegt, dat de MSX terug moet gaan naar nummer 10:

```
20 GOTO 10
```

We kennen de PRINT-opdracht uit hoofdstuk 1. Er is iets bijgekomen: een nummer. Dit nummer is het regelnummer. Zo'n regelnummer staat ook voor de tweede opdracht in het programma. Als een opdracht begint met een regelnummer weet de MSX dat deze opdracht niet direct uitgevoerd moet worden, maar dat hij de opdrachten moet "opsparen" tot het programma compleet is.

Dat opsparen doet de MSX altijd als een regel met een getal begint.
Regels die niet met een nummer beginnen worden direct uitgevoerd, regels die wel met een nummer beginnen horen bij een programma en worden niet direct uitgevoerd.
De regelnummers vertellen de MSX nog meer. Hij weet daardoor ook in welke volgorde de opdrachten uitgevoerd moeten worden. De MSX begint altijd met het laagste nummer en gaat dan verder naar boven.

De regelnummers mogen we zelf kiezen. We mogen beginnen met 10 en met stappen van 10 omhoog gaan, maar ook met 1 of 5 of 100. We moeten er alleen aan denken dat als we met 1 omhoog gaan er geen opdrachten meer tussengevoegd kunnen worden. Meestal gebruiken we stappen van 10 omhoog zoals in het voorbeeld.



19. NEW

Voordat we de MSX een programma geven moeten we eerst zorgen dat het geheugen van de computer leeg is. Als we dat niet doen blijft het oude programma in het geheugen staan. Daar kunnen we last van hebben. We kennen al twee manieren om het geheugen leeg te maken. De eerste is de MSX uit en weer aan zetten, de tweede is indrukken van de RESET-toets.
Nu komt er een derde manier bij. We kunnen het geheugen ook leeg maken met de opdracht: NEW. Maak eerst het scherm schoon met CLR (SHIFT-HOME).

Wij: **new** (denk aan RETURN)
MSX: **Ok**
Wij: **nieuw**
MSX: **Syntax error**
Ok

Opmerking: we zullen van hier af de Ok's zoveel mogelijk weglaten. Ok betekent altijd dat wij weer aan de beurt zijn om een nieuwe opdracht te geven. Denk er ook aan, dat we elke opdracht of regel moeten afsluiten met RETURN en dat we kunnen verbeteren, zolang we RETURN nog niet hebben gebruikt.

Nu gaan we de MSX het programma geven. Op het tv-scherm staat als laatste: Ok. Wij typen de volgende twee regels. Elke regel sluiten we af met RETURN:

Wij: **10 print "Carmen"**
20 goto 10

Dan zit het programma nu in het geheugen. De MSX heeft nog niets uitgevoerd. Hij heeft alleen het programma in ontvangst genomen en wacht tot wij zeggen dat hij moet doen wat er in het programma staat. Daardoor komt het ook dat de MSX nog niet kan vertellen of er in onze opdrachten fouten zitten. Er kan dus nog geen syntax error op het scherm staan.

20. LIST

Eerst willen we wel eens weten of het programma nu echt in het geheugen van de MSX zit. Daar kunnen we achter komen met de opdracht **list.**

Geef de MSX deze opdracht. Let op: vóór de opdracht LIST staat géén regelnummer. Deze opdracht hoort niet bij het programma en de MSX moet hem onmiddellijk uitvoeren. Als we de opdracht LIST goed gegeven hebben staat er nu op het scherm:

Wij: **list**
MSX: **10 PRINT "Carmen"**
20 GOTO 10

Ok

De Ok betekent weer dat wij aan de beurt zijn om een opdracht te geven. De MSX heeft de LIST-opdracht uitgevoerd. Nu doen we het volgende:

a. Vergelijk de opdrachten die onder de LIST op het scherm staan met de opdrachten die we moesten typen. Als ergens een fout zit typen we de regel met de fout gewoon opnieuw op de plaats waar de cursor staat. De MSX zorgt er wel voor dat de verbeterde regel op de juiste plaats in het programma komt te staan. Zorg er wel voor dat het goede regelnummer ervoor staat. De regel die niet fout is laten we staan. Dan geven we opnieuw een opdracht LIST. Doe dit net zo lang tot de opdrachten op het scherm precies kloppen met de opdrachten in het boek.

b. De MSX heeft de opdrachtwoorden (PRINT en GOTO) in hoofdletters gezet. Carmen (tussen aanhalingstekens) is gewoon Carmen gebleven. Opdrachten mogen in hoofd- en kleine letters getypt worden. De MSX maakt er altijd hoofdletters van.

c. Als we er helemaal niet uit kunnen komen beginnen we gewoon opnieuw: maak het scherm schoon, geef een opdracht NEW en typ het programma opnieuw in. Als we zeker weten, dat het programma goed is kunnen we verder gaan. Het programma zit nu in het geheugen van de MSX, maar de MSX heeft nog steeds niets gedaan. In het programma staat wel wát hij moet doen maar we hebben hem nog niet verteld dát hij dat nu ook maar eens moet uitvoeren. Dat doen we met de opdracht RUN.

21. RUN en STOP

Geef de MSX deze RUN-opdracht en kijk wat er gebeurt:

Wij: **run** (denk aan RETURN)

a. Op het tv-scherm verschijnt de naam Carmen in een lange rij. De onderste regel met Carmen staat te bibberen en te trillen. In dat geval hebben we goed geprogrammeerd.
b. Op het scherm staat "Syntax error". Er zit dan een fout in het programma.

Als het programma goed werkt gaan we in deze paragraaf verder. Als er een fout in zit gaan we direct naar de volgende.

Het programma werkt dus goed. Op het tv-scherm staat een kolom met de naam Carmen. De MSX voert steeds de opdracht om de naam Carmen op het scherm te zetten uit. Elke PRINT-opdracht zorgt dat er een Carmen bij komt onderaan op het scherm. Alle andere Carmen's schuiven dan een regel naar boven. Onderaan komt er steeds een regel bij, bovenaan gaat er een af (dat heet "scrollen"). Dat gaat alleen zo snel, dat we het niet eens goed kunnen zien!

Nu moeten we de MSX maar eens laten stoppen met zijn werk. Dat doet hij niet vanzelf: het programma zegt dat hij steeds opnieuw de naam Carmen op het scherm moet zetten en daar gaat hij "eeuwig" mee door, tot we de machine uitzetten of op RESET drukken. We kunnen de MSX laten stoppen met de STOP-toets.

Wij: **STOP** (de toets, niet het woord STOP!)



Het witte vierkante blokje, de cursor blijft stilstaan, maar de MSX is nog niet "echt" opgehouden met zijn werk. Hij gaat weer verder als we STOP nog een keer indrukken. Probeer dat:

Wij: **STOP**
MSX: **Carmen**
Carmen (enzovoorts!)

Stop de MSX nog een keer met STOP.

MSX: **Carmen**
Carmen (enzovoorts)
Wij: **STOP** (toets)
MSX: (de MSX "pauzeert")

De positiewijzer staat stil, de MSX "pauzeert". Om de machine echt te laten stoppen met het uitvoeren van een programma, hebben we de toets CTRL nodig.

CTRL is en afkorting van "ConTRoL"
het Engelse woord voor "besturen"
of "besturing".



Deze control-toets CTRL werkt net als de SHIFT: hij moet ingedrukt en vastgehouden worden. Daarna slaan we een andere toets aan en laten CTRL weer los.

Om de MSX te laten stoppen moeten we CTRL-STOP typen. Dat wil zeggen: druk CTRL in, houd hem vast, sla STOP aan, laat CTRL los:

Wij: **CTRL-STOP**
MSX: **Break in 10**
Ok

Break in 10 betekent dat de MSX het programma afgebroken (break) heeft, terwijl hij bezig was met de opdracht op regel 10.

We kunnen een programma dus onderbreken met STOP: de MSX pauzeert en gaat weer verder als we nog een keer op STOP drukken.

We kunnen een programma afbreken met CTRL-STOP. Aan de mededeling Break in ... kunnen we zien waar de MSX in het programma bezig was. De Ok op het scherm geeft aan dat de computer wacht op een nieuwe opdracht. Dat kan bijvoorbeeld een opdracht LIST of een nieuwe RUN zijn. Probeer ze beide.

Opmerking: om een programma af te breken hoeven we niet eerst te pauzeren. Direct CTRL-STOP werkt ook.

22. Fouten in programmaregels

We zullen met een leeg geheugen en een schoon scherm beginnen.

Wij: **CLR** (of SHIFT-HOME)
new
10 plint "Carmen"
20 goro 1

Er zitten drie fouten in dit programma: er staat PLINT in plaats van PRINT, er staat GORO in plaats van GOTO en er staat 1 in plaats van 10.

We gaan deze fouten <u>niet</u> verbeteren maar bekijken wat de MSX daarvan vindt. Staat het programma klaar? Geef dan de opdracht:

Wij: **run**
MSX: **Syntax error in 10**
Ok

Syntax error in 10 betekent dat er in regel 10 een fout zit, waardoor de MSX de opdracht niet begrijpt. We weten natuurlijk welke fout dat is.

We gaan deze foutieve regel nu verbeteren. Dat doen we door de regel opnieuw te typen, mét het regelnummer. Dus:

Wij: **10 print "Carmen"**

Bekijk met de opdracht **list** hoe het programma er uitziet. Het resultaat is:

MSX: **10 PRINT "Carmen"**
20 GORO 1

Is regel 10 nu goed? Zo niet dan moeten we hem nog maar een keer verbeteren, door de regel mét het regelnummer nog een keer (maar dan goed) te typen. Regel 20 laten we nog even fout staan. Staat nu het programma op het scherm zoals we dat willen? Dan geven we nog een keer de RUN-opdracht:

Wij: **run**
MSX: **Carmen**
Syntax error in 20
Ok

De MSX heeft de opdracht PRINT één keer uitgevoerd: de naam Carmen staat op het scherm. Op regel 20 staat iets wat hij niet begrijpt. Daarom staat er "Syntax error in 20".

Hoe verbeteren we? Door regel 20 mét het regelnummer opnieuw te typen:

Wij : 20 goto 1

Die foute 1 laten we nog steeds staan. Controleer eerst even of het programma nu zo in het geheugen staat als we willen.

```
Wij: list
MSX: 10 PRINT "Carmen"
     20 GOTO 1
     Ok
```

En nu proberen wat het resultaat is:

```
Wij: run
MSX: Carmen
     Undefined line number in 20
     Ok
```

De MSX heeft de eerste opdracht uitgevoerd. De naam Carmen staat er. Bij de tweede opdracht op regel 20 heeft hij geen syntax error ontdekt omdat hij begrijpt wat hij moet doen. Maar er is wel iets anders aan de hand. De "Undefined line number" betekent dat we de MSX verder willen laten gaan met een regelnummer, waarbij geen opdracht hoort. Dat komt omdat we GOTO 1 getypt hebben in plaats van GOTO 10. Regel 1 komt in ons programma niet voor. Die fout moeten we verbeteren. We kunnen dat direct doen door de goede regel 20 te typen, maar vaak weten we niet wat er nu precies op die verkeerde regel staat. Dat willen we dus wel eens zien.

```
Wij: list
MSX: 10 PRINT "Carmen"
     20 GOTO 1
```

Als we alléén regel 20 willen zien kan dat met dezelfde LIST-opdracht, maar dan met een regelnummer erin (niet ervoor!):

```
Wij: list 20
MSX: 20 GOTO 1
```

Een opdracht LIST met een regelnummer erachter geeft ons dus één regel uit het programma. Nu kunnen we die regel opnieuw typen:

```
Wij: 20 GOTO 10
```

Tussen twee haakjes: vergeet niet het regelnummer ervoor te zetten. Controleer dan of het programma goed in het geheugen staat met:

```
Wij: list
MSX: 10 PRINT "Carmen"
     20 GOTO 10
```

Geef de opdracht waarmee we het programma uitvoeren:

```
Wij: run
MSX: Carmen
     Carmen
     Carmen
     enzovoorts!
Wij: CTRL-STOP
MSX: Break in 10
```

23. Gemak dient de programmeur

Op de laatste regel van het BASIC-scherm staan vijf woorden:

color auto goto list run

Druk de toets SHIFT in en houd die vast. Kijk dan naar de laatste schermregel:

color cload" cont list. run

Deze tien woorden zijn de betekenissen van de tien "functietoetsen" van de MSX:



De functoetsen F1 tot en met F5 kunnen we gewoon aanslaan (nog niet doen!), voor F6 tot en met F10 hebben we de toets SHIFT nodig op de bekende manier. Met deze functietoetsen kunnen we ons het typen op de MSX gemakkelijker maken: met één functietoets krijgen we een heel woord op het scherm. Welke woorden dat zijn zien we op de onderste schermregel:

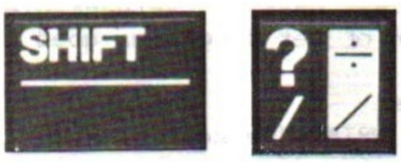
color	auto	goto	list	run
F1	F2	F3	F4	F5

en met SHIFT:

color 15,4,7	cload"	cont	list.	run
F6	F7	F8	F9	F10

We zullen een aantal van deze toetsen proberen en omdat we toch bezig zijn het ons gemakkelijk te maken, nemen we er meteen nog een bij: het woord print kunnen we typen met een vraagteken.

Als we in een opdracht **?** typen maakt de MSX daar het woord PRINT van. Dat zien we pas als we het programma bekijken met een LIST-opdracht.



Proberen:

Wij: **new**
CLR
10 ? "Carmen" (vergeet RETURN-toets niet)
F4 (de functietoets en RETURN)
MSX: **list**

De cursor blijft achter het woord list staan. Waarom? Omdat in de opdracht LIST nog een regelnummer ingevuld moet worden. Dat hebben we in de vorige paragraaf juist ontdekt (bijvoorbeeld list 20 om alleen regel 20 te krijgen).

We willen nu het hele programma zien (dat is toch maar één regel), dus slaan we de RETURN aan:

Wij: **RETURN**
MSX: **10 PRINT "Carmen"**
Ok

Met functietoets F4 krijgen we dus de opdracht list op het scherm. We kunnen nog een regelnummer invullen en sluiten af met RETURN.

In de programmaregel op het scherm is het vraagteken een PRINT-opdracht geworden. We programmeren verder:

Wij: **20 F3** (de functietoets)
MSX: **20 goto**

De cursor staat weer te wachten achter het woord goto. We vullen de opdracht verder in:

Wij: **10 RETURN** (de functietoets voor LIST)
MSX: **10 PRINT "Carmen"**
20 GOTO 10

Funtietoets F5 is de "run-toets". Druk erop als er geen fouten in de twee programmaregels staan:

Wij: **F5** (RETURN is niet nodig!)
MSX: **Carmen**
Carmen
Carmen enzovoorts
Wij: **CTRL-STOP**
MSX: **Break in 10**
Ok

Functietoets F10 is ook een "run-toets", maar deze functie maakt eerst het scherm schoon, voordat de MSX met de uitvoering van het programma begint.

Wij: **F10** (SHIFT-F5, RETURN is niet nodig)
MSX: scherm schoon
Carmen
Carmen
Carmen enzovoorts
Wij: **CTRL-STOP**
MSX: **Break in 10**
Ok

Opmerking: het scherm schoonmaken gaat zo snel, dat we het de eerste keer misschien niet eens zien. Probeer het gerust een paar keer.

We kunnen nu nog één toets uit het rijtje functietoetsen proberen: F8, de toets "cont" van "continue". Dat is Engels voor "ga door".
De opdract CONT kunnen we gebruiken om de MSX verder te laten gaan met het uitvoeren van een programma na een "Break in ..."

MSX: **Break in 10**
     Ok
Wij: **F8**   (SHIFT-F3, RETURN is niet nodig)
MSX: **Carmen**
     Carmen
     Carmen enzovoorts
Wij: **CTRL-STOP**
MSX: **Break in 10**
     Ok

We kunnen zelf kiezen of we de functietoetsen wel of niet willen gebruiken. Alles wat met de functietoetsen kan, kunnen we ook gewoon als opdrachtwoord typen. Maar zoals gezegd in de titel van deze paragraaf: gemak dient de programmeur. Het is alleen wel even wennen! De andere functietoetsen komen later aan de beurt.

Opmerking: de tekst die bij een functietoets hoort past niet altijd helemaal op het scherm. F6 geeft color 15,4,7, maar alleen het woord color staat op de laatste schermregel.

24. Anders printen

We gaan het programma nog eens bekijken. Haal het weer op het scherm of typ het opnieuw als het niet meer in het geheugen zit:

Wij: **list** (mag met F4 RETURN)
MSX: **10 PRINT "Carmen"**
20 GOTO 10

Nu gaan we een kleine verandering aanbrengen in het programma. Typ regel 10 opnieuw en vergeet niet het regelnummer ook te typen.

Wij : **10 PRINT "Carmen";**



Achter de opdracht PRINT staat nu een puntkomma. Wat doet deze puntkomma? Daar kunnen we gemakkelijk achter komen als we de MSX het programma laten uitvoeren:

Wij: **run** (mag met F5 of F10)
MSX: **CarmenCarmenCarmen** (enzovoorts, een scherm vol!)

Wat is het verschil tussen PRINT "Carmen" en PRINT "Carmen";? Bij de eerste opdracht komt de naam in één kolom, bij de tweede opdracht zet de MSX de naam achter elkaar op het scherm. De puntkomma zorgt ervoor dat de MSX bij een PRINT-opdracht niet op een nieuwe regel begint.

Wij: **CTRL-STOP**
MSX: **Break in 10**
Wij: **list** (mag met F4 RETURN)
MSX: **10 PRINT "Carmen";**
20 GOTO 10

LET OP! vanaf dit punt laten we in dit boek de preciese aanwijzingen over het geven van de opdrachten RUN en LIST en over het stoppen (afbreken) van een programma zoveel mogelijk weg. We zeggen gewoon: start het programma, bekijk het programma, stop het programma of breek het programma af.

Daarom nog even een korte samenvatting:

```
start het programma        : typ run RETURN of ...
                             typ F5 (zonder RETURN) of ...
                             typ F10 (zonder RETURN)

bekijken                   : typ de opdracht list RETURN of ...
                             typ F4 RETURN

bekijken van één regel     : typ list met het regelnummer RETURN of ...
                             typ F4, het regelnummer en RETURN

afbreken (stoppen) van
het programma              : typ CTRL-STOP
```

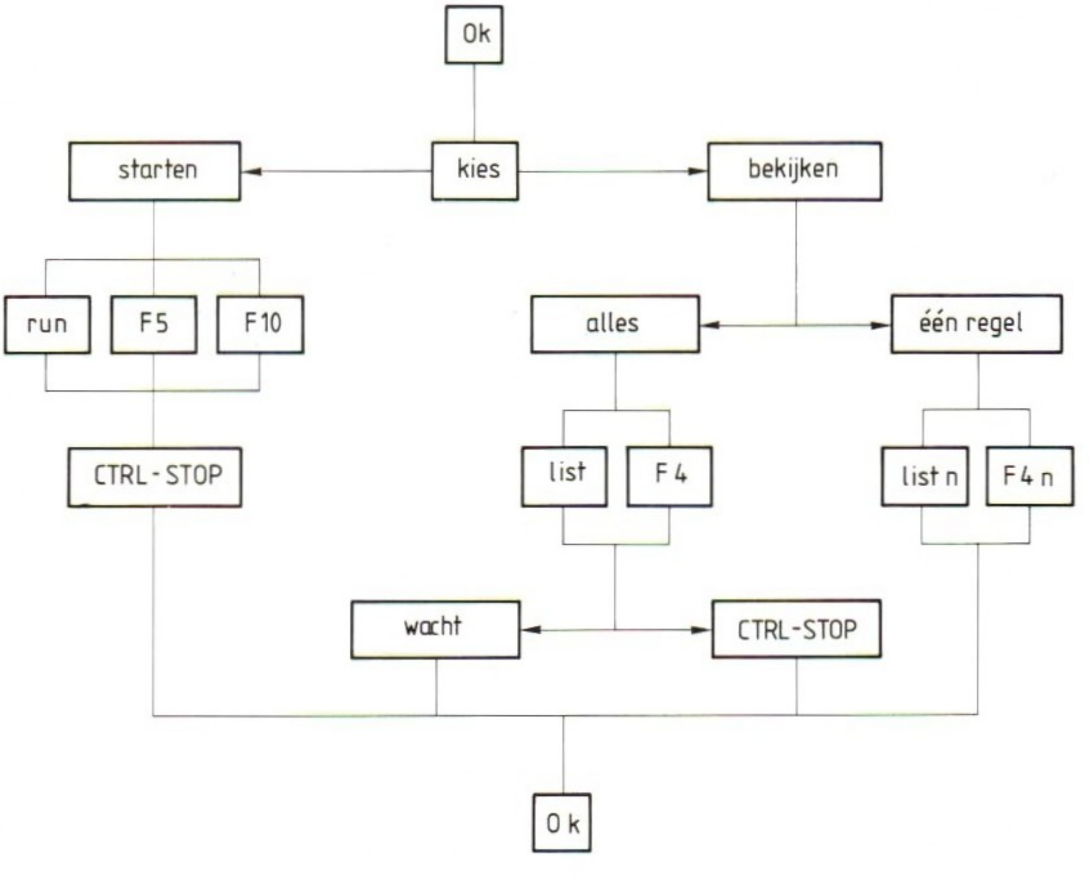
Experimenteer met het programma. Vervang regel 10 door:

Wij: **10 print "Carmen ";** (één spatie tussen n en ")

Voer het programma uit en stop het weer. Vervang nu de PRINT-opdracht op regel 10 door één van de volgende opdrachten. Voer het programma uit, stop het weer en probeer een volgende regel 10.

```
Wij: 10 print "Dory    ";      (vier spaties)
     10 print "Dory        "; (acht spaties)
     10 print "D o r y ";      (steeds één spatie)
```

Denk eraan dat het programma pas werkt als we een RUN-opdracht gegeven hebben.



HOOFDSTUK 4: Strings

25. Stringvariabelen

In de MSX zit een geheugen. Daarin onthoudt de computer wat hij doen moet. We kunnen ons het geheugen het beste voorstellen als een verzameling vakjes. In elk "vakje" kunnen we informatie opbergen. Er zijn twee soorten informatie.

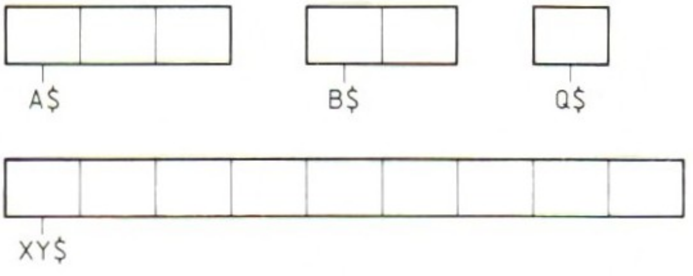
a. Informatie die de computer vertelt wát hij doen moet. Dat zijn de regels met opdrachten van het programma. De "vakjes" met opdrachten geven we een nummer: het regelnummer. Door het nummer van het vakje op te geven kunnen we de computer iets over dat vak vertellen. Bijvoorbeeld dat we in vak 10 (regel 10) een andere opdracht willen zetten.
b. Informatie waarméé we de computer iets willen laten doen. Zulke vakken hebben we nog niet gebruikt. Deze vakken krijgen geen nummer maar een "naam".

De vakken in het geheugen van de MSX kunnen groot of klein zijn. Dat wil zeggen: in sommige vakken komen veel tekens (letters, cijfers en andere tekens), in andere maar weinig. De vakken die we het eerst gaan gebruiken krijgen een naam die bestaat uit één of twee letters en een dollarteken. Bijvoorbeeld: we zien hier vakken met de namen A$, B$, Q$ en XY$

De naam moet met een letter beginnen. De naam van een vak mag uit meer letters (en ook cijfers) bestaan. Voor de MSX zijn alleen de eerste twee tekens (twee letters of letter en cijfer) belangrijk. De twee namen: ANS$ en ANT$ mogen we allebei gebruiken, als we er maar aan denken dat de MSX alleen naar de eerste twee letters kijkt. ANS$ en ANT$ zijn dus voor de MSX precies hetzelfde: de MSX "ziet" alleen AN$! Het dollarteken moet achter al deze namen staan.

Opdracht 1:
Kijk naar de volgende rij namen:

A$	NUL$	ALSJEBLIEFT$
AB$	OPA$	BAL$
AN$	POP	A1$
BA$	QUIZ$	JIF$
JA$	JOZEF$	BAR$
QR$	4Q$	PLAN$



LET OP! Het maakt niets uit of we hoofd- of kleine letters gebruiken in deze namen. De MSX maakt er altijd hoofdletters van.

In deze namen zijn er drie hetzelfde en twee zijn er fout. Welke zijn dat?

26. Strings in vakjes

De vakken met de namen die we hier gebruikt hebben zijn geschikt om strings op te bergen. Strings zijn teksten. In de PRINT-opdracht hebben we de string steeds tussen aanhalingstekens gezet. Daardoor was hij goed te herkennen.

We gaan de MSX nu vertellen dat hij strings in vakjes moet opbergen. Maak het geheugen van de MSX leeg met de opdracht NEW. Maak ook het scherm schoon.

Wij: **a$="Carmen"**

Wat zegt de MSX? Zegt hij alleen maar Ok, dan is het oké. Het kan ook zijn dat op het tv-scherm staat: **Type mismatch.** De MSX is dan in de war geraakt. Dat gebeurt bijvoorbeeld als we het eerste aanhalingsteken hebben weggelaten:

Wij: **a$=Carmen"**
MSX: **Type mismatch**



De computer verwacht na A$= een tekst (b.v. een naam), die met een " begint. Als we het eerste aanhalingsteken weglaten "zegt" hij: "Hier klopt iets niet!".

Als we typen: **a$="Carmen** is er niets aan de hand en begrijpt de MSX ons wel.

We hebben de MSX nu verteld dat hij de naam Carmen moet opbergen in vakje A$. Hij heeft dat ook gedaan, maar daar kunnen we niets van zien. Als we willen weten of hij zijn werk goed heeft gedaan geven we de volgende opdracht:

Wij: **print a$** (of... ? a$, want ? wordt PRINT)
MSX: **Carmen**

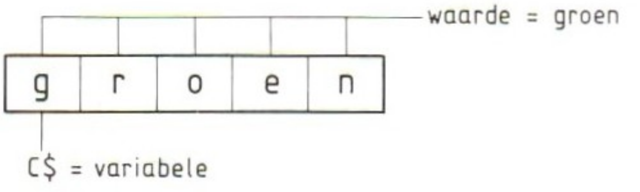
De tekst in vakje a$ komt nu tevoorschijn.

Opmerking: namen van vakken mogen we met kleine of met hoofdletters typen. De MSX maakt er altijd hoofdletters van.

27. Overschrijven

Als de MSX de naam uit A$ op het scherm zet betekent dat niet dat die naam uit vakje A$ verdwenen is. Dat kunnen we zien als we de volgende opdrachten proberen:

Wij: **print a$**
MSX: **Carmen**



Wij: **c$="groen"**
Wij: **print c$**
MSX: **groen**

Wij: **print a$**
MSX: **Carmen**

En jawel, wat in A$ stond staat er nog steeds. Maar nu gaan we dit proberen:

Wij: **a$="gras"**
Wij: **print a$**
MSX: **gras**

Carmen is uit A$ verdwenen en gras staat er nu in. In een vak blijft altijd staan wat we erin gezet hebben behalve als we:

a. het geheugen leeg maken met uit zetten, RESET of NEW.
b. in dat vak iets anders zetten.

Als iets uit een vak verdwijnt doordat er iets anders in gezet wordt heet dat overschrijven (met de klemtoon op schrijven). In dit geval kunnen we zeggen: "Carmen is overschreven door gras".
De vakken in het geheugen heten geheugenvelden, velden of variabelen. Een geheugenveld of variabele noemen we bij zijn naam. Wat er in het geheugenveld of de variabele staat noemen we de inhoud of de waarde.

Wat we nu aan het doen zijn is inhouden in velden zetten of anders gezegd: waarden toekennen aan variabelen. En om het nog wat preciezer te zeggen: in de laatste opdrachten hebben we waarden toegekend aan stringvariabelen.

28. Meer printen

Typ de volgende opdracht:

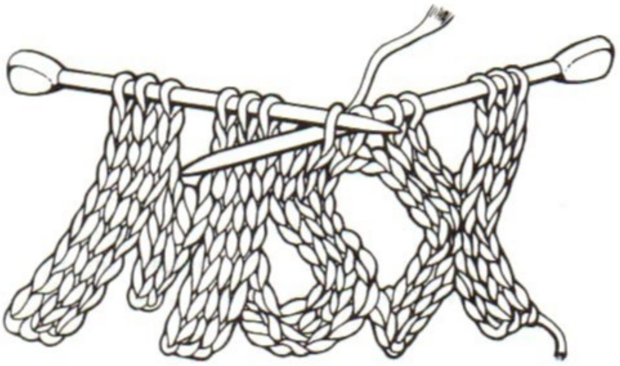
Wij: **print a$ "is" c$**
MSX: **grasisgroen**

Met de PRINT-opdracht kunnen we meer dingen tegelijk afdrukken. In deze PRINT-opdracht staan twee variabelen (A$ en C$) en één waarde ("is"). Die drie dingen worden door één PRINT-opdracht afgedrukt. Maar... het staat er niet zo als we zouden willen. Gras, is en groen zijn aan elkaar op het scherm gezet. Het zou mooier zijn als tussen gras en is en tussen is en groen een spatie staat. Dat kunnen we zo voor elkaar krijgen:

Wij: **print a$ " is " c$** (voor en na is een spatie, binnen de aanhalings-tekens!)

De MSX zet nooit iets anders op het scherm dan waarden. Wélke waarden hij op het scherm moet zetten kunnen we aangeven door in de PRINT-opdracht een variabele te noemen of door de waarde zélf in de PRINT-opdracht te zetten. In dit voorbeeld hebben we twee waarden genoemd door de variabele in de PRINT-opdracht te nemen. De MSX zet niet A$ op het scherm maar de waarde die bij deze variabele hoort. Op het scherm staat ook niet de variabele C$ maar de waarde die op dit moment bij deze variabele hoort.

Tijd voor twee experimenten.



a. Wij: **ans$="MSX"**
anneke$="tv-toestel"
print antwerpen$

b. Wij: **p$="breiwerkje"**
q$=p$
print q$
Wij: **print p$**

Uit deze experimenten leren we:

a. dat namen die met dezelfde twee letters beginnen voor de MSX op dezelfde variabele slaan. ANS$ is dus overschreven door ANNEKE$ en als we ANTWERPEN$ afdrukken komt de waarde die we aan ANNEKE$ toegekend hebben te voorschijn.
b. we kunnen de MSX vertellen dat de waarde die in het ene vak staat ook in een ander vak gezet moet worden. In P$ hebben we een breiwerkje gezet maar door de opdracht Q$=P$ komt datzelfde breiwerkje ook in Q$ te staan.

29. Anton is fout

Nu zullen we kennismaken met een klein probleem bij het gebruiken van namen voor variabelen. Probeer de volgende opdracht:

Wij: **anton$="Anton"**
MSX: **Syntax error**

De opdracht is heel duidelijk, maar toch zegt de MSX dat hij hem niet snapt. Als we het anders proberen gaat het wel goed:

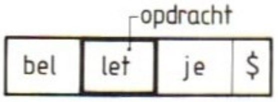
Wij: **annemarie$="Anton"**
MSX: **Ok**

De MSX vindt dit oké. We kunnen dus zeggen dat het probleem niet zit in de ANTON tussen aanhalingstekens, dus niet in de string.

In de variabelenaam ANTON$ zit een woord dat de MSX al kent in een bepaalde betekenis. Dat woord is TO. De MSX kent TO uit een opdracht, die wij later leren. Als we dus als opdracht geven: ANTON$="ANTON" "denkt" de MSX dat er staat: AN TO N$="ANTON" en daarvan zegt hij: "dit begrijp ik niet." We kunnen daarvoor een regel geven: in de naam van een variabele mag nooit een woord zitten dat de MSX als opdrachtwoord kent. Kijk hoe de MSX reageert op:

Wij: **eprintebal$="eprintebal"**

Wij: **eprinsebal$="eprintebal"**



Wij: **belletje$="belletje"**

Wij: **bessentje$="belletje"**

In de eerste opdracht zit PRINT en dat gaat niet. In de derde opdracht zit LET en dat kan ook niet. LET is een opdracht, die we nooit gebruiken, omdat hij weggelaten mag worden. Hij mag staan in b.v.: LET A$="GRAS" (is hetzelfde als A$="GRAS" zonder LET).

Er zijn veel BASIC-opdrachten, die we nog niet kennen en die we daarom per ongeluk in namen zouden kunnen gebruiken. Wees gewaarschuwd! In het BASIC-boekje bij de MSX staan deze zogenaamde "gereserveerde woorden" allemaal.

30. GOTO waarheen?

We keren nu terug naar het programma waarmee we dit hoofdstuk begonnen zijn. Het oude programma zag er zo uit:

```
10 PRINT "Carmen"
20 GOTO 10
```

Dat programma staat niet meer in het geheugen. Zet het er opnieuw in.

Wij: **CLR**
new
10 a$="Carmen"
20 print a$;
30 goto (nog géén RETURN gebruiken!!)

Hier stoppen we even. Regel 30 is nog niet af, maar dat komt direct. Op regel 10 staat dat de string Carmen toegekend moet worden aan de stringvariabele A$. Dat zal de MSX bij de uitvoering van het programma dus het eerst doen. Regel 20 zegt dat de string die toegekend is aan de variabele A$ afgedrukt moet worden. De puntkomma vertelt hem dat hij niet naar een nieuwe regel moet gaan. Regel 30 zegt waar de MSX verder moet gaan. Achter GOTO komt nog een regelnummer. Wat moet hier staan: **30 GOTO 10** of **30 GOTO 20** ???

Als we het antwoord niet weten kunnen we het het beste maar proberen.

Wij: **30 goto 10**

Voer het programma uit met RUN en kijk wat er gebeurt. Stop het programma en typ regel 30 opnieuw:

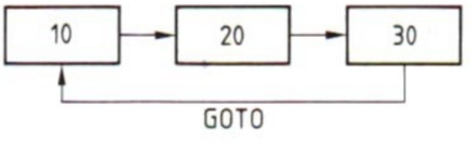
Wij: **30 goto 20**

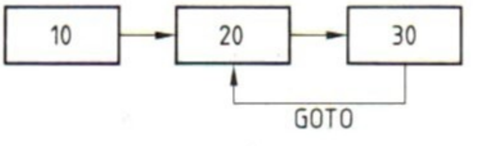
Geef weer een RUN en bekijk het resultaat.

Hier komt alles nog eens op een rijtje:

Wij: **10 a$="Carmen"**
20 print a$;
30 goto 10
run
MSX: **CarmenCarmenCarmen** enzovoorts
Wij: **CTRL-STOP**
MSX: **Break in 20**
Ok
Wij: **30 goto 20**
run
MSX: **CarmenCarmenCarmen** enzovoorts

Als we geen fouten hebben gemaakt kunnen we het verschil op het tv-scherm niet zien. De twee programma's werken precies hetzelfde. Toch kunnen we zeggen dat regel 30 GOTO 20 beter is dan regel 30 GOTO 10, want als we dit laatste kiezen wordt de string Carmen steeds opnieuw in vak A$ gezet.
En we weten intussen dat dat niet nodig is. Als dat één keer op regel 10 gedaan is blijft in vak A$ de waarde Carmen staan. De inhoud van A$ wordt niet overschreven.






```
10 A$="Carmen"
20 PRINT A$
30 GOTO 10
```

Hier doet de MSX méér dan nodig is.

```
10 A$="Carmen"
20 PRINT A$
30 GOTO 20
```

Dat is dus beter.

Opmerking: vanaf hier zullen we programmaregels in dit boek zo schrijven als ze na een opdracht list op het MSX-scherm komen. In hoofdletters dus, als de MSX er zelf hoofdletters van maakt.
We weten inmiddels dat we MSX-BASIC-opdrachten in kleine letters mogen typen.

31. Groen gras ...

Verander nu regel 10 van het programma. Zet een andere tekst tussen de aanhalingstekens en kijk naar het effect van één of meer spaties.
Bijvoorbeeld zo:

```
10 A$="MSX "  (één spatie)
10 A$="MSX  " (twee spaties)
10 A$="MSX   " (drie spaties)
10 A$="MSX    " (vier spaties)
```



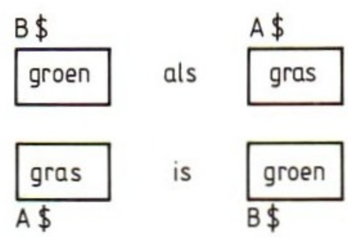
We kunnen natuurlijk maar één van deze vier voorbeelden tegelijk uitproberen.
Vergeet niet het regelnummer te typen en start het progamma met RUN.

Het volgende programma mag geen problemen meer geven. Lukt het niet in één keer? Geen nood, gewoon de verkeerde regel opnieuw typen en als het helemaal mis gaat opnieuw beginnen met een schoon scherm en de opdracht NEW.

```
Wij: 10 A$="gras"
     20 B$="groen"
     30 PRINT B$ " als " A$ (denk aan de spaties)
     40 PRINT A$ " is " B$  (denk aan de spaties)
     50 GOTO 30
     run
MSX: groen als gras
     gras is groen
     groen als gras
     gras is enzovoorts!
Wij: CTRL-STOP
MSX: Break in (30 of 40, dat hangt ervan af, wanneer we precies stoppen)
     Ok
```

Roep het programma nu op met LIST:

```
Wij: LIST
MSX: 10 A$="gras"
     20 B$="groen"
     30 PRINT B$ " als " A$
     40 PRINT A$ " is " B$
     50 GOTO 30
```



Kijk het programma na en breng zo nodig verbeteringen aan. We doen dat nog steeds door een verkeerde regel opnieuw te typen met hetzelfde regelnummer.

32. ... en wat daarmee fout kan gaan

Als er fouten in het programma zitten kunnen we door de MSX op de vingers getikt worden met één van de volgende berichten:

SYNTAX ERROR: : fout in een opdracht. Typ de goede opdracht, met het regelnummer.

TYPE MISMATCH : een " of een $ vergeten. Bekijk de foute regel met LIST met daarachter het regelnummer. Typ de regel opnieuw, met het regelnummer.

UNDEFINED LINE NUMBER: in de opdracht GOTO staat een regelnummer, dat niet in het programma voorkomt. Typ de regel gewoon opnieuw, met een regelnummer vooraan en achteraan.

Opmerking: als achter GOTO géén regelnummer staat geeft de MSX een syntax error.

In een programma kunnen ook fouten zitten die de MSX niet herkent. Hij begrijpt de opdracht dan wel en doet ook wat we gezegd hebben. Maar wat wij in het programma hebben gezet is niet wat we eigenlijk willen. In dit programma is dat bijvoorbeeld het geval als er een cijfer 0 afgedrukt wordt. In één van de PRINT-opdrachten hebben we dan een dollarteken of een aanhalingsteken vergeten. Zoek met LIST uit welke regel dat is en typ die regel opnieuw met het regelnummer.

Als het programma goed werkt willen we toch die berichten over fouten in het programma eens zien. Typ het volgende programma. Er zit een hele reeks fouten in.

```
Wij: NEW
     10 A$=gras"              (een " vergeten: type mismatch)
     20 B="groen"             ($ vergeten: type mismatch)
     30 PLINT B$ als A$       (syntax error)
     40 PRINT A " is " B$     (0 op het scherm)
     50 GOTO 70               (undefined line number)
```

Probeer RUN en LIST.

Als we veel teksten gebruiken lopen we het risico dat de MSX met het volgende bericht komt:

MSX: **Out of string space in ...**

De MSX heeft een "werkruimte" voor 200 tekens. Als die ruimte vol is krijgen we dit bericht. We kunnen de MSX meer ruimte geven met de opdracht:

Wij: **CLEAR 500** (500 tekens)

Nu is het niet zo, dat alle strings meetellen. We kunnen daarom rustig wachten tot we dit bericht een keer tegenkomen.

We kunnen het bericht oproepen door b.v. strings steeds langer te maken (in een later hoofdstuk komen we daarop terug):

```
Wij: NEW
     10 A$="A"
     20 A$=A$+"A"    (We plakken er een A aan vast!)
     30 PRINT A$
     40 GOTO 20
     RUN
```

Met een opdracht CLEAR kunnen we dit probleem oplossen. Probeer achtereenvolgens b.v.:

```
Wij: 5 CLEAR 1000    (1000 tekens werkruimte)
```

Opmerking: een string kan niet langer worden dan 255 tekens. Als dat dreigt te gebeuren krijgen we het bericht: String too long in ...

Een regel die we in het programma willen <u>invoegen</u> typen we gewoon met een regelnummer, dat ervoor zorgt dat de opdracht komt op de plaats waar we hem hebben willen. Door in dit programma regelnummer 5 te kiezen zorgen we ervoor dat de CLEAR-opdracht vooraan in het programma komt te staan, nog vóór regel 10.

```
Wij: LIST
MSX: 5 CLEAR 1000
     10 A$="A"
     20 A$=A$+"A"
     30 PRINT A$
     40 GOTO 20
Wij: RUN
MSX: AAA  enzovoort
     String too long in 20
```

De string is nu 255 tekens groot. Typ ? A$ en tel maar na!



33. Opdrachten

Opdracht 2:

Namen mogen niet beginnen met een cijfer. Wat gebeurt er als we typen:

```
Wij: NEW
     6H$="WIM"
     PRINT 6H$
```

Opdracht 3:

Maak een programmaatje, waarmee we het hele scherm vullen met de tekst GRAS IS GROEN*GROEN IS GRAS*GRAS IS GROEN*enzovoorts!
In het programma mag maar één PRINT-opdracht staan en de woorden GRAS en GROEN mogen maar één keer gebruikt worden in het programma!

Opdracht 4:

Haal alle fouten uit het volgende programma:

```
10 A$="HANS
20 B$=GRIET"
30 PLINT AS EN BS "IS":
40 PRINT B$ "JE" EN A$ "JE"
50 GOTO 15
```

HOOFDSTUK 5 : Editten

34. Editten

In het eerste hoofdstuk hebben we de mogelijkheden van het MSX-toetsenbord verkend. We hebben daarbij kennis gemaakt met een aantal toetsen, dat gebruikt kan worden bij het verbeteren van fout getypte opdrachten en teksten.

Dat waren: BS (backspace)
INS (insert, invoegen)
DEL (delete, weghalen)
de cursortoetsen

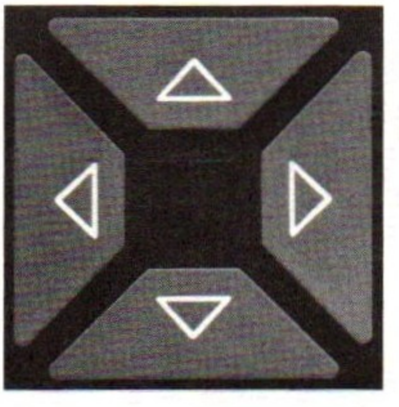
Daarna hebben we het verbeteren een beetje vergeten: regels met fouten erin of regels die we gewoon wilden veranderen hebben we helemaal opnieuw getypt. Daar moet nu maar eens een eind aan komen: we gaan "editten", d.w.z. veranderingen aanbrengen in een programma zonder alles opnieuw te typen.

LET OP! Doe vooral de eerst bladzijden precies wat in dit boek staat. Zelf experimenteren mag straks, maar als we daar nu al aan mee beginnen zijn we binnen de kortste keren de draad kwijt!

De MSX heeft een zogenaamde "full screen editor". Dat wil zeggen dat we overal op het scherm kunnen komen met de cursor om iets wat al op het scherm staat te veranderen.

Typ eerst een klein programma:

```
Wij: CLR
     NEW
     10 PRINT "Het is donderdag vandaag"
     20 GOTO 10
     RUN
MSX: Het is donderdag vandaag
     Het is donderdag vandaag
     enzovoorts
Wij: CTRL-STOP
MSX: Break in 10
     Ok
Wij: LIST
MSX: 10 PRINT "Het is donderdag vandaag"
     20 GOTO 10
     Ok
```



Loop nu met de cursortoetsen tot de cursor op de eerste d van donderdag staat (_ geeft de plaats van de cursor aan):

Dit is dus invoegen

Typ nu:

Wij: **woens**
MSX: **10 PRINT "Het is vandaag woensrdag"**
Wij: **DEL** (de DELete-toets)
MSX: **10 PRINT "Het is vandaag woensdag"**
Wij: **RETURN**
MSX: **20 GOTO 10**

We hebben nu regel 10 van ons programma "ge-edit". d.w.z. met behulp van de edit-mogelijkheden van de MSX veranderd. Dat gaat zo:

1. We gaan met de cursortoetsen naar de plaats waar we de veranderingen willen aanbrengen.
2. Met behulp van de toetsen INS, DEL en BS en door over de oude woorden heen te typen kunnen we veranderingen aanbrengen.
3. Met de toets RETURN geven we het sein: verander het nu ook maar in het geheugen. Dat is heel belangrijk: de RETURN zorgt ervoor dat de veranderingen ook écht in het geheugen komen. Als we de RETURN vergeten verandert er wel iets op het scherm, maar niet in het programma zoals het in het geheugen staat!

Intussen staat de cursor aan het begin van regel 20. De MSX verwacht dat wij op deze regel ook iets gaan veranderen. Dat is niet de bedoeling.
We kunnen nu twee dingen doen: óf we lopen met de cursortoetsen van deze regel weg tot we weer "in de vrije ruimte" komen (op een regel waar nog niets staat), óf we typen RETURN. Als we deze laatste mogelijkheid kiezen, zeggen we in feite tegen onze MSX: het veranderen van deze regel (regel 20) is klaar! Omdat we nog niets veranderd hebben, blijft de regel zoals hij was.

Probeer het eerst zo:

Wij: loop met de cursor naar de eerstvolgende vrije regel
list (mag nog steeds met F4 RETURN)
MSX: **10 PRINT "het is woensdag vandaag"**
20 GOTO 10
Ok

De donderdag is dus inderdaad veranderd in een woensdag.

Laat het programma nog even in het geheugen staan! Het mag ook uitgevoerd en weer gestopt worden.

35. Invoegen

Maak eerst de zaak wat overzichtelijker.

```
Wij: CLR
     list
MSX: 10 PRINT "Het is woensdag vandaag"
     20 GOTO 10
     Ok
Wij: 15 PRINT "Hoera!"
     list
MSX: 10 PRINT "Het is woensdag vandaag"
     15 PRINT "Hoera!"
     20 GOTO 10
     Ok
```

Een hele regel invoegen gaat zo: kies een regelnummer dat ligt tussen de twee regelnummers vóór en ná de nieuwe regel. Typ de in te voegen regel met dit regelnummer. De MSX zorgt ervoor dat de regel op de juiste plaats terecht komt.

Loop nu met de cursortoetsen naar de H van Hoera!:

```
MSX: 15 PRINT "Hoera!"
Wij: INS  (de toets INSert)
     Vrij!
MSX: 15 PRINT "Vrij!Hoera!"
Wij: spatie (met de spatiebalk van de MSX)
MSX: 15 PRINT "Vrij! Hoera!"
Wij: RETURN
MSX: 20 GOTO 10
```

Iets vergeten in een regel? Geen nood. Loop erheen, druk op INS en voeg in wat vergeten is. Let wel op de vorm van de cursor: invoegen doen we zolang de cursor een streepje is. Als het weer een blokje wordt (na een cursortoets, na HOME, RETURN of CTRL-STOP) is het invoegen afgelopen en typen we weer over de oude opdrachten en teksten heen.

Loop met de cursor naar de eerste lege regel en start het programma. Genoeg gezien? Dan stoppen.

```
Wij: CLR
     list
MSX: 10 PRINT "Het is woensdag vandaag"
     15 PRINT "Vrij! Hoera!"
     20 GOTO 10
     Ok
```

We gaan nu in regel 10 de volgende veranderingen aanbrengen:

1. Tussen in en woensdag voegen we in: alweer
2. Achter woensdag zetten we: middag
3. Vandaag halen we weg.

Dit karweitje kunnen we met INS, DEL en BS op verschillende manieren doen. Dit is dus maar een voorbeeld:

Wij: loop naar het laatste aanhalingsteken op regel 10
MSX: **10 PRINT "Het is woensdag vandaag"**
Wij: druk 8 keer op **BS**
MSX: **10 PRINT "Het is woensdag"**
Wij: **INS**
middag
MSX: **10 PRINT "Het is woensdagmiddag"**
Wij: loop met de cursortoetsen naar de w van woensdag (de cursor wordt weer een blokje; het invoegen is beëindigd)
MSX: **10 PRINT "Het is woensdagmiddag"**
Wij: **INS**
alweer (en een spatie erachter)
MSX: **10 PRINT "Het is alweer woensdagmidda**
g"

Loop nu met de cursor naar de eerste lege regel op het scherm.

Wij: **RETURN**
list
MSX: ...

Jawel hoor: het programma staat er nog precies zo als vóór dit lastige edit-karwei! Hoe komt dat? Omdat we niet op RETURN gedrukt hebben toen we nog met regel 10 bezig waren. dat hebben we pas op de lege schermregel gedaan. En dan blijft alles zoals het was!

Hieruit kunnen we twee dingen leren:
1. Als we echt iets op een bepaalde regel veranderd willen hebben, moeten we op die regel het editten afsluiten met RETURN.
2. Als we bij het editten de tel kwijt zijn geraakt en we willen opnieuw beginnen, drukken we niet op RETURN maar "lopen weg" met de cursor naar de eerste de beste lege regel en typen daar onze nieuwe opdracht.

36. Mogelijkheden en fouten bij het editten

Maak het scherm (CLR) en het geheugen (NEW) schoon.

```
Wij: 10 PRINT "Carmen"
     run
MSX: Carmen
     Ok
```

Loop nu met de cursor naar de C van de laatste Carmen (de Carmen die de MSX "geprint" heeft):

LET OP



```
MSX: Carmen
     Ok
Wij: INS
     20  (en een spatie erachter)
MSX: 20 Carmen
Wij: ?   (en een spatie erachter)
MSX: 20 ? Carmen
Wij: " RETURN
MSX: 20 ? "Carmen
     Ok
```

Wat hebben we gedaan? We hebben van een tekst (de naam Carmen) die door de MSX op het scherm gezet was een programmaregel gemaakt. Kijk zelf maar:

```
MSX: Ok
Wij: list  (gewoon over Ok heentypen en natuurlijk RETURN niet vergeten)
MSX: 10 PRINT "Carmen"
     20 PRINT "Carmen  (hier ontbreekt de laatste ", maar dat is niet erg)
     Ok
Wij: run
MSX: Carmen
     Carmen
     Ok
```

Conclusie: we kunnen <u>alles</u> wat op het scherm staat gebruiken bij het editten. Alles? En de laatste schermregel dan (die waar color auto enz staat)?

Op het scherm staat op de tweede regel het woord run. Loop daarheen met de cursor. Druk op RETURN. Wat gebeurt er?
De MSX voert de RUN-opdracht uit: door naar de regel waar deze opdracht staat te gaan en vervolgens op RETURN te drukken, nemen we deze regel (en dus de opdracht run) ongewijzigd over. Maar omdat deze opdracht zomaar ergens op het scherm staat kan het best zijn dat het niet helemaal duidelijk is wat er nu precies gebeurt! Dat komt ook doordat de MSX de regels waarop hij print niet eerst netjes schoonmaakt. Op het scherm kunnen dus vreemde dingen komen zoals:

```
CarmenCarmen
Okrmen
```

Probeer nu zelf: loop over het scherm naar een regel waar iets staat. Druk op RETURN. Kijk hoe de MSX reageert. LET OP: de reactie van de computer komt direct onder de regel waarop we op RETURN gedrukt hebben.

Als we dit een paar keer doen, blijven er alleen nog "syntax errors" op het scherm over. Hoe komt dat?
Probeer dat als volgt: maak het scherm en het geheugen schoon.

Wij: **10 PRINT "I   Dory"**

Voor het hartje hebben we
de toets GRAPH erbij nodig.

LET OP: als GRAPH een omschakeltoets is moet hij ook weer uitgeschakeld worden!

Wij: **list**
MSX: **10 PRINT "I   Dory"**
Ok

Ga nu met de cursor op de O van Ok staan en druk op RETURN:

MSX: **Ok**
Wij: **RETURN**
MSX: Ok
Syntax error
Ok

Waarom deze syntax error? Syntax error betekent altijd: ik (MSX) heb een opdracht gekregen die ik niet begrijp. Welke opdracht is dat?

We zijn met de cursor op de regel met Ok gaan staan. Daar hebben we op RETURN gedrukt en daarmee hebben we de MSX de opdracht gegeven, die op die regel stond.

Op deze regel stond niets anders dan Ok en de MSX heeft dus "gedacht": ze willen dat ik de opdracht Ok uitvoer. Maar... die opdracht ken ik niet. Syntax error!

Geef nu een opdracht RUN

Wij: **run**
MSX: I **Dory**
Ok

Loop nu met de cursor terug naar de regel waarop de opdracht run staat en ga helemaal naar rechts op deze regel. Pas op: Niet naar een andere regel gaan!

Druk op RETURN. Wat gebeurt er?
Schijnbaar gebeurt er niets: de cursor springt weer onder de laatste Ok op het scherm. Maar in werkelijkheid doet de MSX wat hem opgedragen wordt: hij voert de opdracht run weer uit.
We zien daar niets van omdat het resultaat van deze RUN-opdracht op precies dezelfde plaats komt als de vorige keer, en aangezien dat resultaat daar al stond en niet is weggehaald kunnen we niet zien dat de MSX écht weer "geprint" heeft.

Samengevat:
1. We kunnen alles gebruiken wat op het scherm staat.
2. Loop naar de regel waar datgene staat wat we gebruiken willen. Verander het zonodig met BS, INS, DEL en de cursortoetsen. Van een programmaregel kunnen we een directe opdracht maken door het regelnummer weg te halen (DEL of BS). Van een directe opdracht kunnen we een programmaregel maken door een regelnummer in te voegen.
   Een directe opdracht kunnen we opnieuw uitvoeren.
3. Het maakt niet uit waar we op een regel staan. Als we een opdracht willen herhalen moeten we "ergens" op de regel waar deze opdracht staat op RETURN drukken.
4. Als we een regel "overnemen" waar iets anders dan een opdracht staat (bijvoorbeeld Ok) krijgen we een syntax error.

Maak het scherm en het geheugen van de computer schoon en probeer dan het volgende:

Wij: **10 PRINT "ABC"**
20 PRINT "PQR"
loop nu met de cursor naar regel 10 op de A
MSX: **10 PRINT "ABC"**
Wij: **123 RETURN**
MSX: **20 PRINT "PQR"**

We hebben tijdens het programmeren een eerdere regel veranderd (ABC werd 123) en de verandering ook in het MSX-geheugen aangebracht door op RETURN te drukken. De cursor staat nu aan het begin van regel 20. Geef nu een LIST-opdracht:

Wij: **list**
MSX: **list INT "PQR"**

Het woord list staat op de regel waar regel 20 uit het programma stond. De MSX maakt deze regel niet eerst schoon, zodat een deel van de oorspronkelijke regel zichtbaar blijft. Druk nu op RETURN.

Wij: **RETURN**
MSX: **Syntax error**
Ok

Deze syntax error ontstaat doordat de computer de regel: list INT "PQR" als één opdracht ziet en hem dus niet begrijpt!

Geef nu opnieuw een opdracht list. Is regel 20 van het programma nog aanwezig?

37. Regels weghalen

In een eerdere paragraaf hebben we gezien hoe we een regel in het programma kunnen invoegen door een nummer te kiezen tussen twee bestaande nummers.



Wij: **new**
CLR
10 PRINT "Boodschappen"
20 PRINT "een halfje tarwe"
30 PRINT "twee krentenbollen"
15 PRINT "bakker"
list
MSX: **10 PRINT "Boodschappen"**
15 PRINT "bakker"
20 PRINT "een halfje tarwe"
30 PRINT "twee krentenbollen"
Ok

Regel 15 wordt door de MSX keurig tussen regel 10 en regel 20 gezet. Juist om deze mogelijkheid te hebben kiezen we regelnummers die met 10 omhoog lopen.

Wij: **17 PRINT "een slagroomsoesje"**
list
MSX: heeft 17 tussen 15 en 20 geplaatst

Overbodige regels weghalen is nog gemakkelijker: we typen alleen het regelnummer met RETURN en verder niets:

Wij: **list**
MSX: laat regel 10, 15, 17, 20 en 30 zien
Wij: **15 RETURN**
MSX: doet niets (zelfs geen Ok)
Wij: **list**
MSX: laat regel 10, 17, 20 en 30 zien

Regel 15 is verdwenen en komt niet meer in het programma voor.

Wees voorzichtig met het typen van regelnummers: een regelnummer gevolgd door RETURN haalt de regel met dat nummer weg.

Probeer nu ook:

Wij: **18 RETURN**
MSX: **Undefined line number**
Ok

De computer moest regel 18 uit het programma halen, maar... die regel is er helemaal niet. Hij is dus "undefined".

Probeer dan ook dit even:

Wij: **18+7** (RETURN)
MSX: doet niets
Wij: **list**
MSX: laat het programma zien met op regel 18:
18+7

Onze "som" 18+7 is een regel in het programma geworden: regel 18 met als "opdracht" +7. Dat levert straks gegarandeerd een syntax error op!
Haal hem weer weg.

Wij: **18 RETURN**
list

Alles wat met een getal begint is voor de MSX een regel in het programma!

Opmerking: een regel weghalen kan ook met BS of DEL. Ga naar de regel toe en haal de inhoud van de regel weg met de toets BS of met de DEL. Het regelnummer moet blijven staan.
De MSX haalt de lege regel met het nummer uit het programma. Als we ook het nummer weghalen blijft de regel in het programma aanwezig.
Vergeet niet af te sluiten met RETURN.



38. List met een puntje

Het wordt langzaam tijd terug te keren naar het gewone programmeerwerk. Dat doen we in deze paragraaf met nog één edit-mogelijkheid, die erg handig is. De overige mogelijkheden komen in de loop van dit boek aan de orde.
Typ het volgende programma in een leeg geheugen en op een schoon scherm (de fouten staan er met opzet in!):

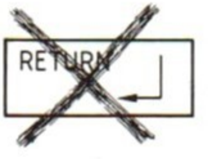
```
Wij: 10 PLRINT "Buiten is het koud"
     20 T$=-6 graden"
     30 PRINT "Het is" T$
     40 GOTO 25
```

De fouten al gevonden? Net doen alsof we niets gezien hebben.

```
Wij: run
MSX: Syntax error in 10
     Ok
```

We moeten nu regel 10 veranderen. Hij staat nog op het scherm, maar we weten inmiddels dat het wel eens lastig kan zijn om overal op het scherm te gaan zitten veranderen. Druk daarom op functietoets F9 (SHIFT-F4):

```
Wij: F9  (RETURN hoeft 'en mag niet)
MSX: list.
     10 PLRINT "Buiten is het koud"
     Ok
```



Functietoets F9 geeft de opdracht LIST. (list met een puntje). We weten dat achter het woord list een regelnummer mag staan, als we maar één regel willen zien.

In plaats van het nummer mogen we ook dat puntje gebruiken. Dat geeft aan dat we de regel willen zien waarbij de MSX is blijven steken.

In dit geval bleef de computer staan bij "Syntax error in 10". Regel 10 is dus de regel waar de MSX is blijven steken. List. laat dus deze regel 10 zien.

De machine doet zelfs meer: hij zet de cursor alvast op deze regel (op het eerste cijfer van het regelnummer) omdat hij ervan uitgaat, dat we deze regel wel zullen willen editten. En dat klopt: de L van PLRINT moet eruit. Breng die verandering aan, druk op RETURN en geef weer een RUN-opdracht.

Wij: **run**
MSX: **Buiten is het koud**
Type mismatch in 20
Ok

In regel 20 staat een fout (" vergeten). De MSX blijft daar steken.

Wij: **list.** (gewoon voluit typen met RETURN)
MSX: **20 T$=-6 graden"**
Ok

Helaas... als we list. voluit typen gaat de cursor niet alvast op de juiste plaats staan! We doen het gewoon over:

Wij: **F9**
MSX: **list.**
20 T$=-6 GRADEN"
Ok

Zet tussen = en - het aanhalingsteken " en druk op RETURN.

Opmerking: doordat we het begin-aanhalingsteken vergeten zijn, heeft de MSX van de hele regel hoofdletters gemaakt. Bij teksten tussen aanhalingstekens doet hij dat niet.

Wij: **run**
MSX: **Buiten is het koud**
Het is -6 graden
Undefined line number in 40
Ok

Een duidelijke fout, nietwaar?

Wij: **F9**
MSX: **list.**
40 GOTO 25
Ok

Regel 40 staat weer klaar om ge-edit te worden. Verander 25 maar in 30. Voer dan het programma opnieuw uit.

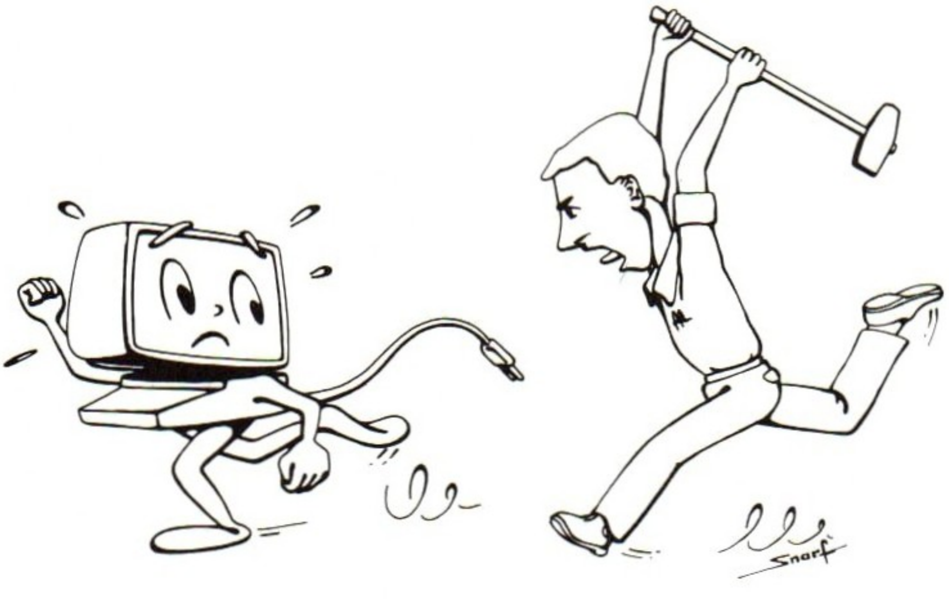
Stop het programma met CTRL-STOP. Hoogst waarschijnlijk krijgen we dan:

MSX: **Break in 30**
Ok
Wij: **F9**
MSX: **list.**
40 GOTO 30
Ok

Een kleine misrekening: we krijgen niet regel 30 te zien, maar regel 40, waarin de GOTO 30 staat.

Daarmee besluiten we dit hoofdstuk over editten. We hebben heel wat mogelijkheden leren kennen en daardoor zal het best wat ingewikkeld zijn geworden.

We kunnen maar één advies geven: blijf het steeds opnieuw proberen, ook al gaat het vaak mis. Het kost nu eenmaal wat tijd om de nodige routine op te bouwen. Af en toe mopperen op de MSX mag ook!



HOOFDSTUK 6: INPUT en de getalvariabelen

In dit hoofdstuk leren we nieuwe opdrachten en mogelijkheden kennen. Vooraf echter een opmerking: we zullen in dit en de volgende hoofdstukken niet meer zeggen dat het scherm en het geheugen schoongemaakt moeten worden.

We weten inmiddels dat een schoon scherm en een leeg geheugen bij het experimenteren wel zo gemakkelijk is. Wees echter niet té snel met schoonmaken: soms hebben we in een volgende paragraaf het programma dat in het geheugen zit nog nodig.

39. INPUT

We beginnen met een nieuwe opdracht, de INPUT. Typ de volgende programmaregel:

Wij: **10 INPUT A$**
     RUN
MSX: ?

Als de MSX anders reageert dan hier is aangegeven zit er waarschijnlijk een fout in regel 10. Geef een opdracht LIST om daar achter te komen en maak de regel correct.
Als de MSX de INPUT-opdracht uitvoert zet hij een vraagteken op het scherm met daarachter de cursor. Verder gebeurt er niets. Het vraagteken betekent dat wij aan de beurt zijn om iets te doen. Doe het volgende:

Wij: **Carmen** (en RETURN)
MSX: **Ok**

Wat is er nu gebeurd? De MSX heeft de INPUT-opdracht op regel 10 uitgevoerd. Deze opdracht zegt dat er iets getypt moet gaan worden op het toetsenbord. Onze opdracht in het programma maakt dat mogelijk. Als de MSX deze opdracht uitvoert zet hij een vraagteken op het scherm en gaat wachten tot we iets ingevoerd hebben. Wij typen Carmen en sluiten deze tekst af met RETURN. Daarmee is de opdracht volledig uitgevoerd.
Omdat de MSX niet méér opdrachten heeft zegt hij Ok en zijn wij weer aan de beurt om iets te doen. De opdracht INPUT gebruiken we dus om het mogelijk te maken dat we tijdens de uitvoering van het programma iets typen op het toetsenbord. Dat heet invoeren of ingeven. We kunnen de INPUT-opdracht het beste interpreteren als "vraag om invoer via het toetsenbord".

Dat "iets" komt ergens in het geheugen van de MSX. In de INPUT-opdracht hebben we gezegd, waar deze tekst opgeborgen moet worden. In dit geval was dat het vakje (variabele) A$. We kunnen dus zeggen dat de opdrachten a en b in het volgende voorbeeld op elkaar lijken.

a. 10 A$="Carmen"   b. 10 INPUT A$

Bij a zorgen we ervoor dat in het vakje A$ de tekst Carmen komt te staan. Bij b zorgen we er met een INPUT-opdracht voor dat er "iets" in vakje A$ komt te staan. We mogen ook zeggen: in geval a leggen we bij het maken van het programma al vast wat we in vakje A$ willen hebben, namelijk de naam Carmen. In geval b leggen we van tevoren nog niet vast wat er precies in A$ moet komen. We maken het mogelijk dat "iets" via het toetsenbord in te voeren. In het volgende voorbeeld kunnen we zien wat daarvan het voordeel is:

```
Wij: 10 INPUT A$
     20 PRINT A$
     30 GOTO 10
     RUN
MSX: ?
Wij: Carmen
MSX: Carmen
     ?
```

We moeten er wel aan denken, dat de MSX steeds bezig blijft met het uitvoeren van het programma. We kunnen als het vraagteken er staat niet zomaar BASIC-opdrachten typen. Probeer dat in het volgende voorbeeld. Als het vraagteken er staat typen we:

```
MSX: ?
Wij: 20 PRINT A$;
MSX: 20 PRINT A$;
     ?
```

Het was de bedoeling om regel 20 van het programma te veranderen door achter de PRINT-opdracht een puntkomma te zetten. Maar wat gebeurt er als we op RETURN gedrukt hebben? Precies: de MSX heeft onze "ingave" in ontvangst genomen op regel 10 en opgeborgen in vakje A$. Dan begint hij aan regel 20 en voert de PRINT-opdracht uit. De inhoud van vakje A$ zet hij op het scherm en daar verschijnt dus de door ons getypte regel weer.

Er staat ook meteen een nieuw vraagteken op het scherm omdat de MSX al weer bij regel 10 is.

We mogen dus zeggen: als het vraagteken er staat is de MSX bezig met het uitvoeren van een programma en op dat moment kunnen we geen nieuwe opdrachten typen of bestaande opdrachten in het programma veranderen.
Als we weer BASIC-opdrachten willen geven moeten we eerst het <u>lopende programma afbreken</u>. We doen dat met CTRL-STOP. De MSX reageert met het bekende:

```
MSX: Break in 10
     Ok
```

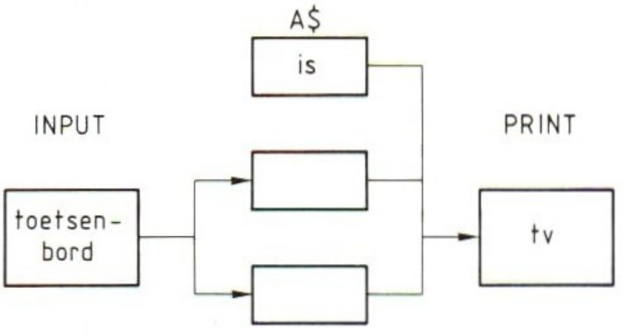
40. Aan de beurt

We kennen nu twee manieren, waarop de MSX ons de beurt kan geven:

a. de MSX heeft niets meer te doen en wacht op nieuwe opdrachten. Dat moeten dan BASIC-opdrachten of programmaregels zijn.
b. de MSX voert een INPUT-opdracht uit. Wij zijn aan de beurt om iets in te voeren op het toetsenbord, maar dat kan geen BASIC-opdracht zijn. Wat we moeten invoeren hangt van het programma af.

Typ het volgende programma:

Wij:
```
10 A$=" is "
20 INPUT B$
30 INPUT C$
40 PRINT B$ A$ C$
50 GOTO 20
```



Kijk even mee naar het programma en controleer meteen of het goed ingevoerd is. Op regel 10 zetten we in vakje A$ de string " is " (met een spatie ervoor en erachter). Op regel 20 zeggen we dat de MSX van ons een string ingevoerd krijgt die hij op moet bergen in vakje B$. Op regel 30 staat weer een INPUT-opdracht. De MSX moet ons hier de kans geven iets in te voeren dat opgeborgen wordt in vakje C$. Op regel 40 moet hij iets op het scherm zetten: eerst de inhoud van B$, dan de inhoud van A$ (daar staat is, dat weten we) en daarachter de inhoud van C$.

Wat er in de vakjes B$ en C$ staat als de MSX aan regel 40 begint leggen we nú nog niet vast: dat laten we afhangen van wat we straks gaan invoeren als de MSX de vraagtekens op het scherm zet.
Op regel 50 gaan we terug naar 20. Dan kunnen we een nieuwe string in B$ en in C$ zetten.
Geef nu een RUN-opdracht. Als het programma niet werkt zoals we hier beschreven hebben zit er een fout in. Breek het programma dan af en verbeter het.

In dit programma hebben we in de PRINT-opdracht op regel 40 drie dingen gecombineerd: twee strings die ingevoerd zijn met INPUT-opdrachten en één string die "vast" in het programma staat. Zo'n string of getal dat "vast" in het programma staat noemen we een constante. In dit programma is de string " is " een constante. Wat in B$ en C$ staat is niet bekend en het is ook niet vast. We kunnen, elke keer als regel 20 en 30 opnieuw uitgevoerd worden, andere strings invoeren. B$ en C$ zijn geen constanten maar variabelen.
Probeer:

Wij: **run**
MSX: **?**
Wij: **blauw**
MSX: **?**
Wij: **groen**
MSX: **blauw is groen**
?

We zien hier dat de MSX precies doet wat wij hem in het programma hebben opgedragen. Hij protesteert niet als hij onzin moet uitkramen. Als we genoeg geëxperimenteerd hebben, breken we het programma af en gaan we verder.

Experimenteer bij het invoeren met o.a.:
- lange teksten
- alleen de toets RETURN
- cijfers
- grafische tekens

41. Lege regels

We gaan nu een wijziging aanbrengen in het programma. Dat kan pas als we het programma afgebroken hebben met CTRL-STOP. Geef dan een LIST-opdracht om te zien hoe het programma er nu uit ziet. Daarna voegen we een opdracht aan het programma toe op regel 35.

Wij: **35 PRINT**

Voer het gewijzigde programma uit. We zien dan dat op het scherm een regel overgeslagen wordt nadat we de tweede tekst hebben ingevoerd. De PRINT-opdracht op regel 35 zorgt daarvoor. Op deze programmaregel staat alleen PRINT en verder niets. Die opdracht gebruiken we om de MSX een regel te laten overslaan op het scherm, als het programma uitgevoerd wordt.

Breek het programma af en voeg toe:

Wij: **25?** (typ hier gewoon een vraagteken)

Op regel 25 staat een PRINT-opdracht.

Als we in een programmaregel een vraagteken typen wordt dit altijd door de MSX omgezet in een PRINT-opdracht, behalve als het vraagteken tussen aanhalingstekens staat. Dan hoort het bij een string en blijft het gewoon een vraagteken. Voer nu eerst het veranderde programma uit en kijk wat het met de nieuwe regel 25 doet: "niets" printen is een regel overslaan.

42. Meer INPUT en wat er fout kan gaan

Typ nu het volgende programma:

Wij: **10 A$=" is " (spaties!)**
20 INPUT B$,C$
30 ?
40 ? A$ B$ C$ "?"
50 GOTO 20

Voordat we het uitvoeren kijken we eerst even wat we precies gedaan hebben. Dit programma lijkt op het vorige maar er zijn drie dingen anders:

a. In de INPUT-opdracht hebben we nu twee variabelen staan. Tussen die twee variabelen staat een komma (niet vergeten!).
b. In plaats van het woord PRINT hebben we een vraagteken getypt. We weten dat dat vanzelf PRINT-opdrachten worden.
c. De volgorde van de drie strings in regel 40 is anders dan in het vorige programma. Bovendien is er nog een vierde string (het vraagteken) aan toegevoegd. Dit vraagteken tussen aanhalingstekens is een string en geen PRINT-opdracht.

Kijk hoe het programma werkt. De "vraag" die de MSX op regel 40 op het scherm zet, staat daar niet netjes. Verander het programma zo, dat het resultaat wel goed is.

Kunnen we ook fouten maken in INPUT-opdrachten? Probeer dat met de volgende voorbeelden.

Wij: **INPUT "A$"**

Hoe reageert de MSX? Precies: met een syntax error. Iets anders viel ook niet te verwachten. Ga nu na hoe de MSX reageert op de volgende fouten:

```
INPUT A$;
INPUT A$ B$
INPUT A$="CARMEN"
```

Ook hier verschijnt steeds de syntax error: we zien eerst een vraagteken en we krijgen pas een syntax error nadat we iets getypt (ingevoerd) hebben. Hieraan kunnen we zien, dat de MSX de opdrachten op een programmaregel van links naar rechts uitvoert. Zolang hij daarbij nog geen fouten ontdekt, gaat alles goed. Neem de laatste regel maar als voorbeeld: bij INPUT A$="CARMEN" begrijpt de MSX INPUT A$ nog wel: het vraagteken komt keurig op het scherm. Pas als wij iets ingevoerd hebben (en afgesloten met RETURN) komt hij bij het restant van de opdracht: ="CARMEN" en dat begrijpt hij niet. Daarom komt dan pas de syntax error.

In deze voorbeelden zitten de fouten in de INPUT-opdrachten zelf. De MSX voert ze daarom niet uit maar meldt syntax errors. We gaan nu kijken of we ook fouten kunnen maken bij het typen als de MSX een vraagteken op het scherm heeft gezet, dus tijdens de uitvoering van een INPUT-opdracht. Typ daarvoor het volgende programma:

```
Wij: 10 INPUT A$
     20 PRINT A$
     30 GOTO 10
     RUN
MSX: ?
```

Probeer eerst het volgende:

```
Wij: Robbie en Jolanda
MSX: Robbie en Jolanda
     ?
```

Dat is dus geen probleem. Een tekst kan best in het vakje A$ staan. Nu dit:

```
Wij: 84
MSX: 84
     ?
```

In een string mogen cijfers zitten. Straks zullen we zien dat we met deze cijfers niet zo gemakkelijk kunnen rekenen.

Nu proberen we de volgende. Het is daarbij niet zo belangrijk wat we precies typen als we er maar voor zorgen dat het meer dan 200 letters of cijfers zijn.

```
Wij: druk een toets in (bijvoorbeeld een letter) en houd die toets vast tot
     er zeven regels met dezelfde letter op het scherm staan
     RETURN
MSX: Out of string space in 10
     Ok
```

Dit bericht van de MSX hebben we al eens eerder gehad. De MSX heeft in het geheugen voor alle strings bij elkaar een werkruimte van 200 tekens. We hebben nu een string gemaakt die groter is en er dus niet inpast. De oplossing van dit probleem kennen we ook: met de opdracht CLEAR gaan we de MSX eerst vertellen dat we meer ruimte voor strings willen hebben. Pas het programma aan:

Wij: **5 CLEAR 500**
RUN
MSX: **?**

Probeer nu weer een tekst van meer dan 200 tekens. Nu lukt het wel! Probeer dan:

MSX: **?**
Wij: **kwik, kwek en kwak**
MSX: **?Extra ignored**
kwik
?

De MSX reageert met een vreemd bericht. Wat zou er aan de hand zijn? Probeer eerst deze nog:

Wij: **wilt u even bij mij komen, meneer de vries?**
MSX: **?Extra ignored**
wilt u even bij mij komen
?

De MSX zet de tekst op het scherm tot aan de komma. Wat daarna komt, laat hij achterwege. Dat betekent dus dat we in een string die we bij een INPUT-opdracht typen géén komma kunnen zetten. Extra ignored betekent: ik let niet op wat teveel is getypt. Dat teveel is alles ná de komma. Dat is natuurlijk wel vervelend, maar ook daarvoor is een oplossing:

Wij: **"kwik, kwek en kwak"**

Wat is het resultaat? Kwik, kwek en kwak staat op het scherm met de komma ertussen. Wat niet op het scherm staat zijn de aanhalingstekens die we ervoor en erachter hebben gezet. Betekent dat nu weer, dat in een string die we typen bij een INPUT-opdracht geen aanhalingstekens kunnen zitten? Daar komen we snel achter als we proberen:

Wij: **"kwik" en "kwek" zijn namen**
MSX: **?Redo from start**
?

Redo from start betekent letterlijk: doe het (het invoeren) helemaal opnieuw. De MSX accepteert wat we getypt hebben niet.

Conclusie: ook in een string, die we invoeren bij een INPUT-opdracht mogen géén aanhalingstekens staan. Aanhalingstekens geven alleen begin en eind aan en tellen zelf niet mee. Ze mogen dus ook alleen aan begin en eind staan!

43. Spaties ervoor en erachter

We gaan nu eens kijken wat de MSX met spaties doet. Het programma loopt nog steeds. Het vraagteken staat op het scherm en Wij typen:

Wij: **vier spaties** (begin met vier spaties)

We zien, dat de tekst nog steeds vooraan begint. De vier spaties die wij getypt hebben zijn verdwenen. Nu zetten we de spaties achteraan.

Wij: **vijf spaties** (vijf spaties erachter, dan RETURN)

We kunnen niet zien of die spaties nu wel of niet op het scherm staan. Daar kunnen we wel iets aan doen door het programma te veranderen. We gaan ervoor zorgen dat er achter de string die we typen bij de INPUT-opdracht nog iets afgedrukt wordt. Breek het programma af en pas het dan zo aan:

Wij: **LIST**
MSX: **10 INPUT A$**
20 PRINT A$
30 GOTO 10
Wij: **20 PRINT A$ "!" (gebruik de edit-mogelijkheden!)**
RUN
MSX: **?**

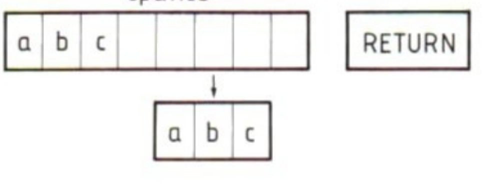
De bedoeling van deze verandering is dat achter de string die we typen door de MSX nog een uitroepteken wordt gezet.

Wij: **abc**

Staat het uitroepteken erachter? Dan is het programma goed. Staat het er niet dan hebben we een fout gemaakt. Breek het programma af, bekijk het en breng de verbetering aan. Als het gelukt is gaan we verder met spaties achter de string:

Wij: **vijf spaties** (vijf spaties erachter, dan RETURN)

Het resultaat is duidelijk. De vijf spaties die we achter de string getypt hebben heeft de MSX ook niet geaccepteerd! Breek het programma af.



44. Overnemen van het scherm

Bij het programmeren kunnen we met de cursor vrij over het scherm bewegen om te "editten" in het programma. Kan dat ook bij het uitvoeren van een INPUT-opdracht?

Probeer het met dit programma:

Wij: **10 PRINT "Carmen"**
20 PRINT "Hoe heet je?"
30 INPUT A$
40 PRINT "Dag " A$
50 GOTO 20

Is de bedoeling van het programma duidelijk? Zet het op een schoon scherm in een leeg geheugen en voer het uit.

MSX: **Carmen**
Hoe heet je?
?

De MSX is nu bij de INPUT-opdracht en vraagt iets in te voeren. Nu typen we niets, maar gaan met de cursor naar de C van Carmen, die al op het scherm staat.

Wij: **Carmen**
RETURN
MSX: **Dag Carmene?**
Hoe heet je?
?

Conclusie: We kunnen iets dat op het scherm staat ook gebruiken als we bij INPUT iets moeten invoeren. Maar of het resultaat ook zo is als we willen? Door op het scherm terug te gaan en Carmen "over te nemen" van het scherm, komt de reactie van de computer op de regel waar al stond:

Hoe heet je?

De MSX voert op die regel nu de PRINT-opdracht PRINT "Dag " A$ uit, maar maakt die regel op het scherm niet eerst schoon. De e van je en het vraagteken blijven dus staan:

Hoe heet je?
Dag Carmen

Probeer nu zelf uit of bij het invoeren bij een INPUT-opdracht ook werken:
a. de toetsen BS, INS en DEL
b. de toets CLR (SHIFT-HOME)
c. de functietoetsen

De conclusie is: alles werkt, maar we moeten blijven bedenken dat de MSX alles wat we typen beschouwt als antwoord op zijn vraag naar INPUT. In dit programma zegt hij dus bijvoorbeeld:

MSX: **Hoe heet je?**
     ?
Wij: **F5**   (functietoets voor run, RETURN hoeft niet)
MSX: **Dag run**

Run wordt pas weer run en list wordt pas weer list als we het programma afgebroken hebben en de MSX met Ok om nieuwe (BASIC-)opdrachten vraagt.

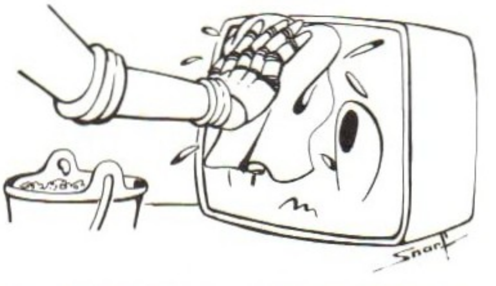
45. Een schoon scherm met CLS

Het wordt nu tijd voor wat nieuws. Breek het programma af en typ de volgende opdracht:

Wij: **CLS**   (van CLear Screen, maak het scherm schoon)
MSX: **Ok**

Als we het goed gedaan hebben is het resultaat duidelijk. De MSX maakt het scherm schoon en begint weer bovenaan.

Maak nu het volgende programma:



Wij:
```
 5 CLEAR 500
10 CLS
20 INPUT A$
30 PRINT "Prettig kennis te maken" A$
40 GOTO 20
```

Regel 10 van dit programma zorgt ervoor dat het scherm door het programma schoongemaakt wordt.
Voer het programma nu uit met steeds een andere naam als de MSX een vraagteken op het scherm zet.
De naam die we typen komt direct achter de tekst Prettig kennis te maken te staan. We gaan regel 30 veranderen (gebruik de edit-mogelijkheden!):

Wij:
```
30 PRINT "Prettig kennis te maken, " A$
35 PRINT
```

We hebben een beetje ruimte gemaakt tussen de string Prettig kennis te maken en de ingevoerde string A$. Bovendien hebben we een blanco regel toegevoegd (programmaregel 35). Voer het programma nu nog eens uit en probeer weer verschillende namen. Breek het programma daarna af.

Opmerking: als we tijdens het programmeren een schoon scherm willen hebben, gebruiken we de toets CLR (SHIFT-HOME). We mogen ook de opdracht CLS (CLear Screen) gebruiken zonder regelnummer ervoor. Als we tijdens de uitvoering van een programma de MSX het scherm willen laten schoonmaken, gebruiken we de opdracht CLS in een programmaregel.

46. Vraag en antwoord

Met de volgende programmaregel maken we van dit programma een "vraag-en-antwoord-spel":

Wij: **15 PRINT "Ik ben de MSX, hoe heet jij?"**
40 GOTO 15
LIST

Als regel 15 toegevoegd is geven we even een LIST-opdracht om het complete programma nog eens te bekijken. Voer het programma daarna uit.
Let op: elke keer als we een RUN-opdracht geven begint de MSX vooraan in het programma. Hij komt dan regel 10 tegen, die zegt dat het scherm schoongemaakt moet worden.

Ook in dit programma zit een schoonheidsfoutje: als de MSX op het scherm zet Ik ben de MSX, hoe heet jij? komen er steeds <u>twee</u> vraagtekens op het scherm. Het eerste vraagteken (achter jij) staat in de string die de MSX op regel 15 moet laten zien. Het tweede vraagteken op de volgende regel komt door de INPUT-opdracht. We kunnen dat veranderen als we het programma zo aanpassen:

Wij: **15 PRINT "Ik ben de MSX, hoe heet jij";**

In de string zit nu geen vraagteken meer. In plaats daarvan hebben we met de puntkomma achter de PRINT-opdracht ervoor gezorgd dat de MSX op dezelfde regel blijft. Probeer het een paar keer uit en breek het programma dan af.
We gaan nu twee veranderingen aanbrengen: regel 15 maken we wat korter en in regel 20 (de INPUT-opdracht) voegen we iets toe:

Wij: **15 PRINT "Ik ben de MSX"**
20 INPUT "Hoe heet jij";A$ (let op de ;!)

Kijk even goed of er geen fouten zijn gemaakt bij het typen van de programmaregels voordat het programma uitgevoerd wordt. Probeer het programma uit en breek het dan af.

In een INPUT-opdracht kunnen we een tekst opnemen, die vóór het vraagteken komt te staan. We moeten er wel goed aan denken, dat achter deze string in de INPUT-opdracht een puntkomma moet staan. Het vraagteken komt automatisch!

Opmerking: op een MSX-scherm passen niet meer dan 37 tekens op één regel. We kunnen daardoor wat vreemde dingen op het scherm krijgen, bijvoorbeeld:

MSX: **Hoe heet jij?**
Wij: **Wilma Winterboer**
MSX: **Prettig kennis te maken, Wilma Winter**
boer

Voorlopig doen we even net of we deze schoonheidsfoutjes niet zien.

47. GOTO en andere problemen

In een vorig hoofdstuk hebben we een keer van de GOTO-opdracht gezegd dat het niet uitmaakt welk regelnummer we daarin zetten. In dit programma maakt dat wel degelijk wat uit.

Bestudeer of onderzoek de gevolgen van verandering van de GOTO-opdracht. Maak er b.v. van: GOTO 10, GOTO 20, GOTO 30, GOTO 35, GOTO 5.

Maak daarna regel 40 weer goed: **40 GOTO 15**

Voer het programma nog een keer uit om te zien hoe het nu werkt. Breek het af en doe het volgende:

```
Wij: LIST
MSX: 5 CLEAR 500
     10 CLS
     15 PRINT "ik ben de MSX"
     20 INPUT "hoe heet jij";A$
     30 PRINT "prettig kennis te maken, "A$
     35 PRINT
     40 GOTO 15
Wij: 35  (en RETURN)
     LIST
```

Regel 35 komt in het programma niet meer voor!

We gaan regel 20 nu een paar keer veranderen. We kunnen dan zien welke fouten we in de INPUT-opdracht beslist niet mogen maken. De bedoeling is dat elke keer regel 20 verandert en het programma daarna uitgevoerd wordt. We krijgen bericht van de MSX over de fout die in het programma zit. Als de computer een fout meldt, komen we met F9 (list.) het snelst bij de regel die we moeten verbeteren.
Hier komen vier voorbeelden van verkeerde INPUT-opdrachten:

a. 20 INPUT "HOE HEET JIJ" A$

b. 20 INPUT HOE HEET JIJ";A$ (Dit is een lastige! De MSX wil een getal om in vakje HOE te zetten!)

c. 20 INPUT "HOE HEET JIJ; A$

d. 20 INPUT "HOE HEET JIJ"; (De fout blijkt pas ná het typen!)

Een verwarrende situatie kan ontstaan als we in het programma een fout maken waardoor we niet kunnen zien dat de MSX opdrachten uitvoert. Typ het volgende programma:

Wij: **NEW**
10 A$ = "niets"
20 GOTO 10
RUN
MSX: (doet "niets"!)

Als we dit programma uitvoeren gebeurt er niets, d.w.z. de MSX is wel aan het werk en voert onze opdrachten uit, maar we zien er niets van. Als we nu iets typen op het toetsenbord gebeurt er helemaal niets. De MSX is bezig met de uitvoering van een programma, maar hij komt geen INPUT-opdracht tegen. Ons typwerk heeft dus geen effect. Het enige dat we kunnen doen is het programma stoppen.
Als we willen zien dat de MSX écht de opdrachten op regel 10 en 20 uitvoert, kunnen we de opdrachten TRON en TROFF gebruiken. Zie daarvoor het BASIC-boekje bij de computer.

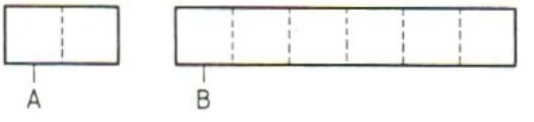


48. Getalvariabelen

We hebben in dit hoofdstuk tot nu toe gewerkt met variabelen en strings of, eenvoudig gezegd, met teksten in vakjes. Nu gaan we hetzelfde doen met getallen. Daarvoor hebben we andere vakjes (variabelen) nodig.
Ook deze variabelen krijgen namen. De regels zijn bijna hetzelfde als bij de variabelen voor teksten: - de naam moet met een letter beginnen - in de naam mag geen BASIC-woord zitten - alleen de eerste twee tekens tellen mee.
Er is één extra regel: achter de naam mag geen dollarteken staan. Dat mag en moet alleen bij variabelen voor teksten.

Opdracht 1:
Zoek in het volgende rijtje welke namen voor getalvariabelen fout zijn en welke namen bij dezelfde variabele horen.

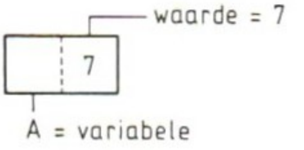
2A	NUL	ALSJEBLIEFT	JAS
AB	OPA	BAL	ANTON
AN	POP	ARIE	BAS
BA	QUIZ	JIF	A$
JA$	JOSEF	BART	AS
QR	QQ	PLAN	A3



Opmerking: getalvariabelen zijn er in drie "maten". Zie daarvoor het BASIC-handboek.

Met getallen in vakjes kunnen we hetzelfde doen als met "gewone getallen". Druk eerst op de RESET-toets, zodat we met een schone MSX kunnen beginnen.

```
Wij: A=7
     PRINT A
MSX: 7
```

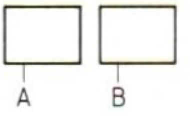


Dit hebben we gedaan:
1. In het vakje met de naam A hebben we de MSX het getal 7 laten zetten.
2. We hebben de MSX opdracht gegeven het getal dat in vakje A staat op het scherm te tonen.
3. De MSX heeft beide opdrachten (direct) uitgevoerd.

We maken nu het geheugen van de MSX weer leeg en gaan proberen of we met getallen in vakjes ook kunnen rekenen.

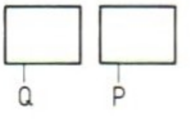
Opmerking: alles mag nog steeds in kleine letters getypt worden.

```
Wij: NEW
     A=7
     B=8
     PRINT A
MSX: 7
Wij: PRINT B
MSX: 8
Wij: PRINT A+B
MSX: 15
Wij: PRINT A-B
MSX: -1
Wij: PRINT A*B
MSX: 56
```



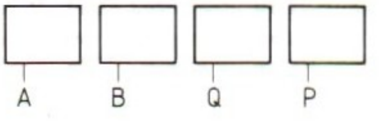
Dat levert dus helemaal geen problemen op. Het rekenen met vakjes gaat precies zo als het rekenen met getallen. We moeten er alleen steeds aan denken dat de letters, hier A en B, betekenen: het getal dat in vakje A en het getal dat in vakje B staat. Net als bij teksten kunnen we getallen in vakjes van het ene naar het andere vak overbrengen. We beginnen met de vakjes Q en P:

```
Wij: NEW
     Q=6
     P=3
     PRINT Q,P   (let op de komma!)
MSX: 6              3
```



In vakje Q staat nu het getal 6, in vakje P het getal 3. De komma in de opdracht PRINT zorgt ervoor dat de 6 en de 3 een eind uit elkaar komen te staan op het scherm. Die komma kunnen we niet weglaten. Als we dat tóch doen staat er PRINT QP (zelfs als we er één of meer spaties tussen zetten!). De MSX denkt dan dat hij iets uit het vakje QP op het scherm moet zetten, en dat vakje bestaat niet. Als we duidelijk willen maken dat P en Q twee vakjes zijn, moet er een komma of een puntkomma tussen.

```
Wij: A=Q
     B=P
     PRINT Q,A
MSX: 6              6
Wij: PRINT P,B
MSX: 3              3
```



Wat hebben we nu gedaan? We zijn begonnen met de opdracht A=Q. Deze opdracht betekent dat de MSX het getal dat in Q staat in A moet zetten. De 6 in Q gaat ook naar A. In Q blijft de 6 staan. Dus: in Q staat 6 en in A staat 6. Ook bij getalvariabelen gelden de regels van het "overschrijven":

Bij een opdracht als A=B wordt A (vóór het = teken) overschreven. B (achter het = teken) wordt niet overschreven. We kunnen ook zeggen: wat in A stond is weg, wat in B stond staat er nog steeds. Alles goed gedaan? Dan wordt het nu tijd om weer eens een fout te maken.

49. Getal of tekst?

We mogen teksten en getallen niet door elkaar gooien. De MSX accepteert dat niet. Als we proberen een tekst in een getalvakje te zetten, waarschuwt hij ons met het bericht TYPE MISMATCH. Dat betekent dat we iets met teksten proberen wat alleen met getallen mag. Andersom geldt dat ook. We kunnen met getallen niet alles doen wat met teksten gedaan kan worden.
Maak eerst het geheugen en het scherm leeg.

Wij: **A="JAN"**
MSX: **Type Mismatch**
Wij: **PRINT A** MSX: ....

Wij: **A$=3** MSX: ....

Wij: **PRINT A$** MSX: ....

We hebben geprobeerd in vakje A de tekst "JAN" te zetten, maar de MSX heeft dat geweigerd. Hij heeft de opdracht niet uitgevoerd en daardoor is in vakje A blijven staan wat er stond. Er staat het getal 0 in, omdat we het geheugen leeg hebben gemaakt. Op de tweede stippellijn komt "Type mismatch". In een tekstvakje (stringvariabele) kunnen we geen getal zetten. We kunnen er wel een string met één of meer cijfers inzetten, maar dan had de opdracht moeten zijn A$="3". De MSX weigert de opdracht A$=3. Daardoor komt er niets in A$. Dat moet dan ook op de derde stippellijn staan: niets (dat is zelfs geen spatie!).

Dit zijn de regels:

a. Achter de naam van een variabele voor getallen staat geen dollarteken. In zo'n variabele kunnen we geen tekst zetten. Doen we dat toch, dan krijgen we het bericht TYPE MISMATCH.
b. Als de MSX een opdracht niet uitvoert omdat hij bijvoorbeeld TYPE MISMATCH ziet, wordt de variabele niet overschreven. Wat er in het vakje stond blijft dus staan.
c. Achter de naam van een variabele voor tekst staat wél een dollarteken. In deze variabele kunnen we wél cijfers in een string zetten. We krijgen dan geen TYPE MISMATCH. Dat krijgen we wel als we een getal (zonder aanhalingstekens) in een tekstvariabele proberen te plaatsen.

We gaan nu de variabelen A$ (voor tekst) en A (voor getal) door elkaar gebruiken. We zullen zien dat dit geen probleem oplevert. A$ en A zijn twee verschillende vakjes. Eén voor tekst, één voor getallen. In de PRINT-opdracht voor het volgende probeersel hebben we geen komma's staan. Kijk zelf wat daarvan het gevolg is.

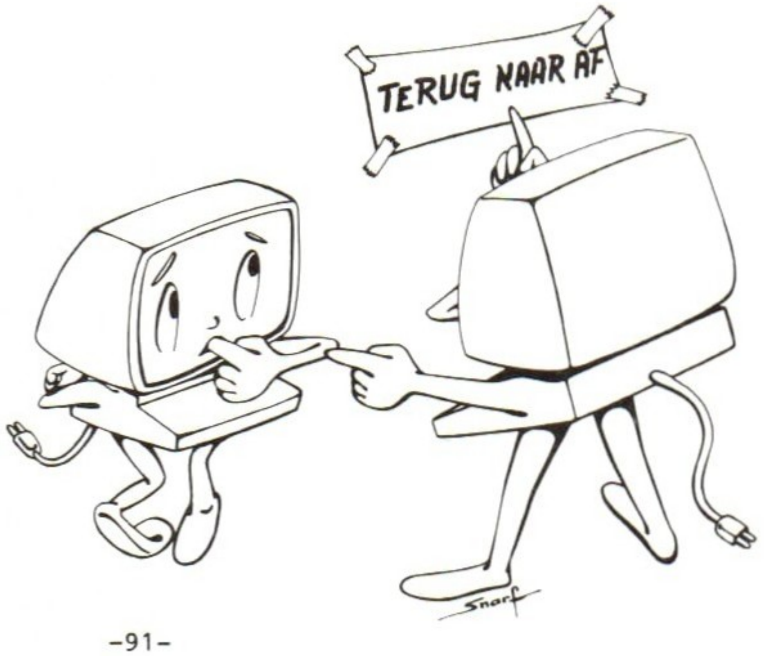
```
Wij: NEW
     A$="jan"
     A=5
     PRINT A$ A  (zonder komma)
MSX: jan 5
Wij: PRINT A$ "is" A "jaar"          MSX: ................................

Wij: PRINT "is" A$ A "jaar?"         MSX: ................................
```

Komma's in PRINT-opdrachten zorgen ervoor dat teksten en getallen een eind uit elkaar komen te staan. Als we alleen met getalvariabelen werken moet er iets tussen omdat de MSX anders in de war komt met de namen. Als we met teksten en getallen door elkaar werken hoeft dat niet altijd. De MSX kan zich niet vergissen als we A$ en A achter elkaar zetten maar weer wél als we A en A$ achter elkaar plaatsen (dat wordt dan AA$!). Bij dit werk moeten we wel steeds opletten of er tussen teksten en getallen wel of niet een spatie komt. Soms moeten we daar zelf voor zorgen door de tekst te laten beginnen of eindigen met een spatie.

We gaan de variabelen voor getallen in een programma gebruiken.
We hebben op regel 10 de INPUT-opdracht gebruikt. Deze opdracht zorgt ervoor dat we een getal kunnen typen dat door de MSX in A gezet wordt.
Daarna drukken we een regel af waarin dit getal voorkomt. Het programma gaat dan terug naar regel 10 waar we een nieuw getal kunnen invoeren. We willen dan ook wel eens zien wat er gebeurt als we geen getal maar een tekst typen.

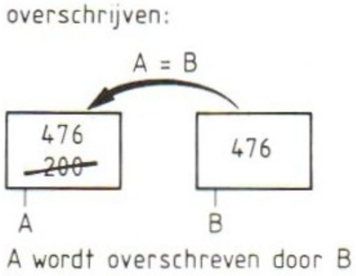
```
Wij: NEW
     10 INPUT A
     20 PRINT "Jan is "  A  " jaar."  (Let op: extra spaties!)
     30 GOTO 10
     RUN
MSX: ?
Wij: 5
MSX: Jan is  5  jaar.
     ?
Wij: nul
MSX: ?Redo from start
     ?
```



De MSX eindigt met een bericht en geeft daarna weer een vraagteken. Op regel 10 moeten we een getal invoeren en geen tekst. Geven we tóch een tekst, dan weigert de MSX die en zegt: doe het nog maar een keer helemaal over. Het vraagteken staat al klaar. Het programma is dus niet afgebroken. Probeer nu uit wat het resultaat van het werk is als we de volgende getallen invoeren:

0
4
24
7E1 (dit is: $7 \times 10^1 = 70$, zie hoofdstuk 2)
-6
HONDERD
"73" (met aanhalingstekens!)
3,7 (met een komma!)
3.7 (met een punt!)

Als het hele rijtje aan de beurt is geweest breken we het programma af met CTRL-STOP (wat gebeurt er als we het woord STOP typen?). In dit rijtje zien we dat we in de INPUT-opdracht alle soorten getallen mogen invoeren. Alleen strings zijn verboden. De teksten HONDERD en "73" kunnen daarom niet. Het komma-probleem in 3,7 is bekend: wat na de komma komt telt niet mee.



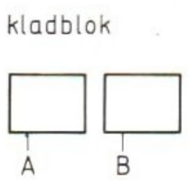
50. Rekenen met getalvariabelen

In het volgende voorbeeld maken we gebruik van de INPUT-opdracht met een tekst. Deze tekst tussen aanhalingstekens komt op het scherm bij het uitvoeren van de INPUT-opdracht. Vergeet vooral de puntkomma niet.
Typ het volgende programma:

```
Wij: NEW  (is het vorige programma al afgebroken)
     10 INPUT "A=";A
     20 INPUT "B=";B
     30 PRINT "A+B=" A+B
     40 PRINT "A-B=" A-B
     50 PRINT "A*B=" A*B
     60 PRINT "A/B=" A/B
     70 PRINT
     80 GOTO 10
```

Geef LIST en controleer of het programma goed in het geheugen staat. Daarna kunnen we het gaan uitvoeren:

```
Wij: RUN
MSX: A=?
Wij: 5
MSX: B=?
Wij: 0
MSX: A+B= 5
     A-B= 5
     A*B= 0
     A/B=
     Division by zero in 60
     Ok
```



Als we het programma goed getypt hebben werkt het uitstekend met de getallen 5 en 0, totdat we bij het delen komen. Daar staat het bericht DIVISION BY ZERO met Ok eronder. Dit betekent dat de MSX het programma afgebroken heeft omdat hij moest delen door nul en dat mag hij niet. De MSX stopt dan bij de deelopdracht (het eerste stuk van de PRINT op regel 60 kon wel!). Hij waarschuwt met een bericht, breekt het prograam af en wacht op een nieuwe opdracht. We kunnen nu het programma nog eens proberen met andere getallen. Geef eerst een RUN-opdracht en probeer kleine en grote getallen, gebroken getallen (gebruik de punt i.p.v. de komma) en negatieve getallen (met het minteken ervóór).

51. Komma en puntkomma

Nog een laatste voorbeeld. In regel 50 van het volgende programma hebben we tussen de variabelen A, B en C een puntkomma gezet in plaats van een komma. Typ het programma en kijk wat het resultaat is.

Wij:
```
NEW
10 INPUT "getal";Q
20 A=Q
30 B=Q+Q
40 C=Q*Q
50 PRINT A;B;C
60 GOTO 10
RUN
```

Door de puntkomma tussen de variabelen A, B en C komen de getallen dicht bij elkaar op het scherm te staan op dezelfde regel. Eerder in dit hoofdstuk hebben we tussen de variabelen een komma gezet. Daardoor werd de ruimte tussen de getallen groot. De puntkomma zorgt voor een kleine tussenruimte.

Verander met de edit-mogelijkheden van de MSX regel 50 om te zien wat het effect is als we tussen A, B en C niets zetten of er komma's tussen plaatsen. Probeer het programma met op regel 50:

```
50 PRINT A B C
50 PRINT A,B,C
50 PRINT A;B;C;
```

Samengevat:

a. Als we tussen getalvariabelen niets zetten komt de MSX in de war: hij beschouwt A B C als één variabele ABC. Dat gaat dus niet.
b. Als we tussen getalvariabelen een komma zetten krijgen we een grote tussenruimte, zo groot zelfs dat er geen drie getallen op één regel passen.
c. Als we tussen getalvariabelen een puntkomma zetten, krijgen we een kleine tussenruimte. De drie getallen passen nu wel op één regel.
d. Als we aan het eind van een PRINT-opdracht een punt of een puntkomma zetten, gaat de volgende PRINT of INPUT-opdracht op dezelfde regel verder.
e. Achter een INPUT-opdracht kan géén puntkomma staan. Na een INPUT-opdracht gaat de MSX dus altijd naar een nieuwe regel.

52. Opdrachten

Opdracht 2:
De komma en de puntkomma hebben een speciale functie in de opdracht PRINT. Probeer uit te vinden wat ze doen in het volgende programma:

```
 10 A$="A"
 20 B$="B"
 30 C$="C"
 40 P=5
 50 Q=33
 60 PRINT P A$ B$ Q C$
 70 PRINT P,A$,B$,Q,C$
 80 PRINT P,A$,B$,Q,C$,
 90 PRINT P;A$;B$;Q;C$
100 PRINT P;A$;B$;Q;C$;
110 PRINT "EINDE"
```

Opdracht 3:
Maak een programma, waarmee de oppervlakte van een rechthoek uitgerekend wordt. De lengte en de breedte moeten we bij een INPUT-opdracht kunnen typen. Er moet tekst bij staan, zodat we weten wat we moeten typen en wat de uitkomst voorstelt.

Opdracht 4:
Wat doet de MSX als hij het volgende programma uitvoert? Wanneer en waarom stopt hij?

```
10 INPUT "GETAL";A
20 PRINT A
30 A=A*A
40 GOTO 20
```

Opdracht 5:
We kunnen de MSX ook laten tellen. Wat zal hij doen als hij het volgende programma uitvoert? Wanneer en waarom stopt hij?

```
10 B=0
20 CLS
30 INPUT B$
40 B=B+1
50 PRINT "DAT WAS" B
60 GOTO 30
```

Wat gebeurt er als we regel 60 veranderen in 60 GOTO 20 of 60 GOTO 10?

HOOFDSTUK 7 : Herhalen met FOR-NEXT

53. Steeds weer Carmen

In de vorige hoofdstukken hebben we al gezien dat één van de mogelijkheden van de computer het herhaald uitvoeren van een stuk programma is.
Tot nu toe hebben we de MSX steeds onbeperkt laten herhalen. We moesten elke keer de MSX stoppen met de STOP-toets of wachten tot er een fout in het programma zat, bijvoorbeeld bij de DIVISION BY ZERO. In dit hoofdstuk leren we een mogelijkheid kennen om de MSX een stuk programma een vastgesteld aantal keren te laten herhalen. Daarna stopt hij "vanzelf".

Voordat we beginnen twee opmerkingen: we zullen niet meer steeds WIJ: en MSX: in onze voorbeelden zetten. Verder laten we ook opdrachten als NEW en RUN achterwege.

Typ het volgende programma en controleer het voordat het uitgevoerd wordt:

```
10 FOR A=1 TO 10
20 PRINT "Carmen"
30 NEXT A
```

Wat er gebeurt kunnen we nog beter zien als we het programma veranderen in:

```
10 FOR A=1 TO 10
20 PRINT A "Carmen"
30 NEXT A
40 PRINT A
```

Bij veranderingen als deze gebruiken we natuurlijk de edit-mogelijkheden van de MSX. Omdat dat nog wel wat lastig zal zijn, hier nog een keer hoe we dat stap voor stap doen:

1. Maak het scherm schoon met CLR (SHIFT-HOME)
2. Geef een opdracht list. Het gemakkelijkst gaat dat met functietoets F4 RETURN.
3. Het programma staat nu op het scherm.
4. Loop met de cursor naar regel 20.
5. Zet de cursor op het aanhalingsteken vóór de C van Carmen.
6. Druk op INS; de cursor wordt een streepje.
7. Typ een A en een spatie, druk op RETURN.
8. De cursor is nu weer een blokje en staat op de 3 van regelnummer 30.
9. Loop met de cursor naar de O van Ok.
10. Typ 40 ? A RETURN.
11. Geef een opdracht LIST met F4 RETURN en kijk of het programma zo in het geheugen van de MSX staat als wij dat willen.

Voer dit programma uit en kijk wat er gebeurt met A. Dat kunnen we zien door regel 20 waar A en de naam Carmen op het scherm gezet worden. Daardoor komt er op het scherm: 1 Carmen 2 Carmen 3 Carmen enzovoort, tot en met 10 Carmen. Helemaal onderaan staat nog een keer 11, maar daar staat geen Carmen achter. Dat komt omdat regel 40 buiten de herhaling valt.

Probeer nu het volgende stukje programma. Vul zelf een getal in. Het moet groter zijn dan 1, maar kleiner dan 500 (anders duurt het te lang).

```
10 FOR A=1 TO ...          (zelf een getal invullen)
20 PRINT "Carmen ";        (let op de spatie)
30 NEXT A
```

Gebruik ook nu weer de edit-mogelijkheden van de computer.
Regel 40 krijgen we weg door te typen:

40 RETURN



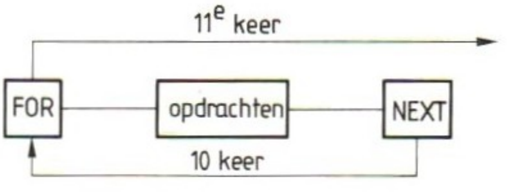
54. FOR-NEXT en wat daarmee kan

Voordat we gaan kijken wat we met FOR en NEXT allemaal kunnen doen, willen we eerst proberen wat wel en wat niet kan. Daarvan een paar voorbeelden. Elk volgend voorbeeld kan door editten uit het vorige gemaakt worden. Geef steeds eerst een LIST-opdracht, voordat we gaan editten. En vergeet niet dat elke "ge-editte" regel afgesloten moet worden met RETURN.

```
10 FOR P=1 TO 10
20 PRINT P
30 NEXT
```

In plaats van NEXT P mogen we ook alleen NEXT zetten. In de voorbeelden in dit boek zullen we de NEXT overal wel invullen, dat is duidelijker. We proberen het met een andere naam:

```
10 FOR JAN=1 TO 10
20 PRINT JAN
30 NEXT JAN
```



En deze:

```
10 FOR A$=1 TO 10
20 PRINT A$
30 NEXT A$
```

In de FOR regel kunnen we geen stringvariabele gebruiken zoals hier A$. Met een stringvariabele kan de MSX niet rekenen en dat moet natuurlijk wel. Hij moet tellen van één tot 10. Daarom geeft hij "Type mismatch".

```
10 FOR W=1 TO 10
20 PRINT W
30 NEXT V
```

We mogen in de NEXT regel wel de letter W weglaten, maar we mogen er niet een andere letter voor in de plaats zetten. Dan raakt de MSX in de war en houdt ermee op: NEXT without FOR betekent, dat de MSX bij deze NEXT-opdracht nog geen FOR-opdracht tegengekomen is.

Probeer:

```
10 FOR W=5 TO 10
20 PRINT W
30 NEXT W
40 PRINT "stop bij" W
```

We laten W beginnen met 5 en doorgaan tot en met 10. Regel 20 wordt dus uitgevoerd met W achtereenvolgens: 5, 6, 7, 8, 9, 10. Dan wordt W gelijk aan 11 en dat is te veel. De MSX stopt met het uitvoeren van de herhaling en gaat verder op de eerste regel na de NEXT-opdracht. Dat is in dit geval regel 40. Hier zet de MSX op het scherm stop bij 11.

Verander regel 10 van dit programma:

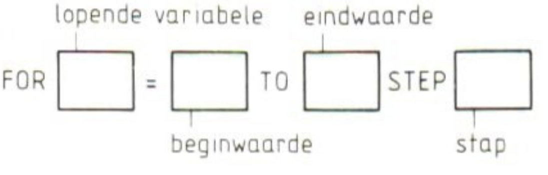
```
10 FOR W=10 TO 5
20 PRINT W
30 NEXT W
40 PRINT "stop bij" W
```

Wat de MSX hiervan zal vinden? Laat hem het programma uitvoeren. Dan zien we meteen hoe de MSX "denkt": hij doet niet moeilijk maar precies wat hem wordt gevraagd. Op regel 10 begint hij met W=10. Op regel 20 laat hij zien wat in W staat: 10. Op regel 30 neemt hij de volgende W. Dat wordt dus 10 + 1 = 11. Daarna gaat hij terug naar regel 10. Daar vraagt hij zich af of hij nog verder moet. Er staat dat hij gaan moet tot en met 5 en W is intussen al 11. Dus moet hij stoppen.
Dat betekent niet meer verder met regel 20, maar naar regel 40. Daar zet hij op het scherm: stop bij 11. We zien dus dat de MSX zich bij de eerste ronde niet afvraagt of hij er al is. Dat doet hij pas bij de tweede en alle volgende ronden. In dit voorbeeld wordt de herhaling dus één keer uitgevoerd en daarna houdt de MSX er direct mee op.

Nog een voorbeeld. Regel 10 verandert, de rest van het programma blijft gelijk.

```
10 FOR W=-1 TO 1
20 PRINT W
30 NEXT W
40 PRINT "stop bij" W
```

Het mag niet moeilijk zijn te verklaren waarom het programma zo werkt.



55. Rekenen en tellen met FOR-NEXT

We kunnen FOR en NEXT gebruiken om ervoor te zorgen dat de MSX een stuk programma een bepaald aantal keren uitvoert.
Dat hebben we in het eerste voorbeeld gedaan. FOR-NEXT zorgde ervoor dat de opdracht 20 PRINT "Carmen" precies 10 keer uitgevoerd werd.
Probeer nu het volgende programma. Het is heel anders dan het programma dat nog in het geheugen staat, dus geef eerst een opdracht NEW.

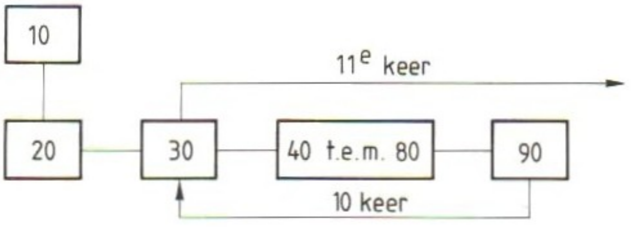
```
10 FOR Q=1 TO 10
20 PRINT Q "*" 2 "=" Q*2
30 NEXT Q
```

Kijk het programma goed na voordat het uitgevoerd wordt. In dit programma gebruiken we een variabele met de naam Q. In dit "vakje" staat een getal waarmee we rekenen. Q is dus een getalvariabele. Met deze Q doen we in dit programma twee dingen:

a. we laten hem lopen van 1 t.e.m. 10 door middel van de FOR-NEXT. We noemen Q daarom de "lopende variabele". D.w.z. Q is een vakje met een getal dat steeds verandert door de FOR-NEXT;
b. in regel 20 gebruiken we Q ook. Hij wordt afgedrukt en er wordt een berekening mee gemaakt. We kunnen dat ook anders zeggen: de lopende variabele wordt gebruikt om het aantal malen dat een herhaling uitgevoerd moet worden te bepalen en kan bovendien in de opdracht(en) in de herhaling zelf gebruikt worden.

In het volgende voorbeeld gebruiken we de lopende variabelen alleen om ervoor te zorgen dat de herhaling op regel 30 t.e.m. 90 tien keer uitgevoerd wordt. De lopende variabele Q wordt in de herhaling zelf niet gebruikt en werkt alleen maar als "teller".

```
10 INPUT "getal";A
20 INPUT "getal";B
30 FOR Q=1 TO 10
40 PRINT "A="A
50 PRINT "B="B
60 PRINT "A*B="A*B
70 A=A+1
80 B=B-1
90 NEXT Q
```



Wat er in dit programma precies gebeurt kunnen we in dit boek niet voorspellen: dat hangt af van de getallen die op regel 10 en 20 tijdens de uitvoering van het programma zijn ingevoerd.
Wat we wel kunnen voorspellen is dat precies tien keer een rekensommetje A maal B is gemaakt met steeds andere getallen. En als dat niet zo is, zit er een fout in het programma.

Typ het volgende programma en probeer het uit.

```
10 CLS
20 PRINT "Carmen";
30 FOR N=1 TO 500
40 NEXT N
50 GOTO 20
```

Merk op dat dit programma weer onderbroken moet worden met de STOP-toets.

In dit programma hebben we een CLS-opdracht gebruikt om het scherm schoon te maken. Die opdracht staat op regel 10. De FOR-NEXT in dit programma zorgt ervoor dat er een poosje "niets" gebeurt. De MSX telt alleen tot 500 voordat hij weer verdergaat. Dat komt omdat er tussen de opdracht FOR op regel 30 en de opdracht NEXT op regel 40 geen andere opdrachten staan. Op deze manier kunnen we de MSX "even laten wachten".
We kunnen natuurlijk ook andere getallen invullen om de MSX korter of langer stil te laten staan. Probeer dat met bijvoorbeeld:

```
30 FOR N=1 TO 1000
```

Typ het volgende programma:

```
10 FOR X=1 TO 20
20 PRINT X
30 NEXT X
40 PRINT "STOP BIJ" X
```

Tot zover niets bijzonders, maar nu gaan we regel 10 veranderen:

```
10 FOR X=1 TO 20 STEP 2
```

Op regel 10 staat nu dat de lopende variabele X van 1 tot 20 moet lopen, maar met stappen van 2 tegelijk. Dat gaat dus zo: X is eerst 1, wordt dan 3, 5, en zo verder tot X gelijk wordt aan 21.
Probeer het programma met deze nieuwe regel 10 uit.

We hebben nog een paar voorbeelden van mogelijkheden voor regel 10:

```
10 FOR X=20 TO 1 STEP -1
10 FOR X=-1 TO -20 STEP -2
10 FOR X=1 TO 100 STEP 10
```

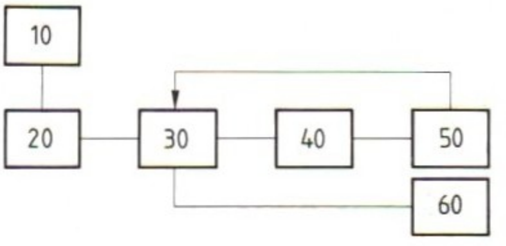
Probeer ze allemaal uit en merk op dat we ook terug kunnen tellen van bijvoorbeeld 20 naar 1 met stapjes van -1. Tel elke keer hoe vaak de opdracht 20 PRINT X uitgevoerd wordt.

Even samenvatten: in de FOR-regel staat a) een lopende variabele, b) een begin, c) een eind, d) een stap. Als de stap 1 is hoeven we dat niet in de FOR-regel te zetten.

56. Variabelen in FOR-NEXT

Tot nog toe hebben we in de FOR-regel steeds constanten gebruikt om aan te geven waar de lopende variabele moet beginnen en waar hij op moet houden. In het volgende voorbeeld gebruiken we daarvoor getalvariabelen. De twee getalvariabelen A en B hebben wél eerst een waarde gekregen.
Probeer:

```
10 A=1
20 B=10
30 FOR X=A TO B
40 PRINT X
50 NEXT X
60 PRINT "stop bij" X
```



Dat werkt dus precies hetzelfde als wanneer we in regel 30 de 1 en de 10 gewoon ingevuld zouden hebben.

```
10 A=1
20 B=10
30 FOR X=A TO B
40 PRINT A;B;X
50 A=A+2
60 B=B-2
70 NEXT X
80 PRINT "stop bij" A; B; X
```

Dit programma werkt ook met de variabelen A en B in de FOR-opdracht op regel 30. Kijk het goed na voordat de RUN-opdracht gegeven wordt. Let vooral op de puntkomma's op regel 40 en regel 80. Voer het programma uit.

Op regel 10 en 20 krijgen de variabelen A en B een waarde. Op regel 30 staat de FOR-opdracht. Die luidt nu: laat X lopen van 1 (de waarde van A) tot en met 10 (de waarde van B). Als de MSX regel 30 uitvoert legt hij dit vast. X moet lopen van 1 t.e.m. 10.

Let op: Zodra de MSX regel 30 uitvoert, legt hij de begin- en de eindwaarde vast. Dat kan niet meer veranderen. Daarna begint de herhaling.

Op regel 40 worden in de eerste ronde de waarden van A en B en de waarde van X op het scherm gezet. Daarna worden A en B op regel 50 en 60 veranderd. Bij A komt er 2 bij, bij B gaat er 2 af. Dat mag, maar bedenk wel dat het geen invloed heeft op de begin- en de eindwaarde die de MSX al vastgelegd heeft. Dat wil dus zeggen dat ook al is A nu 3 geworden en B 8, X blijft lopen van 1 t.e.m. 10. Voor de eerste twee ronden gebeurt er dus het volgende met de waarden van A, B en X:

X moet lopen van 1 tot en met 10

Ronde	Regel	A	B	X
1	40	1	10	1
	50	3	10	1
	60	3	8	1
	70	3	8	2
2	40	3	8	2
	50	5	8	2
	60	5	6	2
	70	5	6	3
3	enzovoorts.			

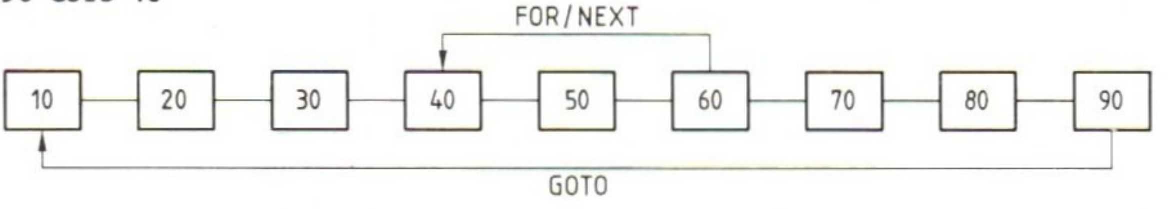
In de 10e ronde worden A en B nog één keer veranderd op regel 50 en 60.
Dan ontdekt de MSX dat hij alle rondjes afgelegd heeft.
Op regel 80 komt dus het volgende:

Ronde	Regel	A	B	X
-	80	21	-10	11

57. Rijtjes maken

FOR en NEXT zijn heel handig om rijtjes te maken. Een bekend voorbeeld van rijtjes zijn de tafels. Typ het volgende programma en probeer het uit.

```
10 INPUT "tafel van";A
20 INPUT "hoeveel keer";B
30 CLS
40 FOR X=1 TO B
50 PRINT X "*" A "=" X*A (het eerste * mag ook x zijn, het tweede niet!)
60 NEXT X
70 PRINT "klaar"
80 PRINT
90 GOTO 10
```



Werkt het programma? Syntax errors mogen er niet in zitten. Die moeten eerst verbeterd worden.

Vraag : Waarvoor dient regel 80?
Antwoord : Deze opdracht zorgt voor een lege regel als de MSX één tafel gemaakt heeft.

Vraag : Hoe kunnen we ervoor zorgen dat het programma met een schoon scherm begint?
Antwoord : We kunnen een regel 5 toevoegen met de volgende opdracht:
5 CLS
Opdracht : Zoek uit waarom het schoonmaken van het scherm in dit programma niet met één opdracht kan.

Vraag : Hoeveel herhalingen zitten er in dit programma?
Antwoord : Twee. Kijk maar naar het schema waarin de regelnummers van dit programma staan. We zien dan dat er twee herhalingen in zitten. Een "grote" die begint op regel 10 en eindigt op regel 90 en een "kleine" met FOR en NEXT van regel 40 t.e.m. regel 60.

Herhalingen kunnen we dus maken met GOTO of met FOR-NEXT. In het volgende voorbeeld maken we de tafels van 1 t.e.m. 10. De FOR-NEXT op regel 80 en 90 zorgt er alleen voor dat elke tafel een poosje blijft staan voordat er een nieuwe op het scherm komt.

```
10 CLS
20 FOR A=1 TO 10
30 PRINT "de tafel van" A
40 FOR B=1 TO 10
50 PRINT B*A;
60 NEXT B
70 PRINT
80 FOR Q=1 TO 5000
90 NEXT Q
100 CLS
110 NEXT A
120 PRINT "klaar"
```

Vraag : Er zijn twee opdrachten die maar één keer uitgevoerd worden. Welke twee zijn dat?
Antwoord : De opdrachten op regel 10 en regel 120

Vraag : Hoe vaak wordt de opdracht op regel 30 uitgevoerd?
Antwoord : 10 keer.

Vraag : Hoe vaak wordt de opdracht op regel 50 uitgevoerd?
Antwoord : 10 x 10 = 100 keer.

Opmerking: dit is een programma met meer herhalingen, dat uit zichzelf stopt.

Het volgende voorbeeld zit iets eenvoudiger in elkaar. Op regel 20 kunnen we een begingetal typen, op regel 30 een eindgetal.
De MSX rekent dan alle kwadraten van het begingetal tot en met het eindgetal uit. Dat gebeurt op regel 50. Dit programma kan vastlopen als we de getallen A en B te groot kiezen.

```
10 CLS
20 INPUT "kwadraten van";A
30 INPUT "tot en met";B
40 FOR M=A TO B
50 PRINT M " kwadraat is " M*M
60 NEXT M
70 PRINT
80 GOTO 20
```


overflow

Vraag : Welke opdracht uit dit programma wordt maar één keer uitgevoerd?
Antwoord : De opdracht op regel 10.

Vraag : Op welke manier stopt dit programma?
Antwoord : Als we het afbreken met CTRL-STOP-toets of als het getal M*M op regel 50 te groot wordt ("Overflow"). Dit getal moet dan wel héél groot worden, namelijk een 1 met 64 nullen.

58. Som en gemiddelde

In deze paragraaf nog een paar voorbeelden van berekeningen met de MSX. We beginnen met het uitrekenen van de som en het gemiddelde van een rij getallen. We laten de MSX eerst vragen om hoeveel getallen het gaat. Dat gebeurt op regel 10. Daarna voeren we de getallen in en tellen ze bij elkaar op. We hebben dan de som gekregen die we op regel 70 op het scherm zetten. Op regel 80 rekenen we het gemiddelde uit. Daarna gaan we terug naar regel 10 voor de volgende ronde. Het programma kan gestopt worden met CTRL-STOP.

```
10 INPUT "hoeveel getallen";N
20 S=0
30 FOR P=1 TO N
40 INPUT A
50 S=S+A
60 NEXT P
70 PRINT "de som is" S
80 PRINT "het gemiddelde is" S/N
90 GOTO 10
```

Vraag : Is de opdracht S=0 op regel 20 nodig?
Antwoord : In de eerste ronde niet. S is dan "vanzelf" 0. Voor alle volgende ronden wel. Elke keer moeten we weer met S=0 beginnen om de juiste som te krijgen. Als er een keer een getal in S staat door de optelling op regel 50 wordt hij niet vanzelf weer 0. We moeten er met een opdracht voor zorgen dat dit wel gebeurt.

Vraag : Waarom hoeft de MSX niet te tellen hoeveel getallen er zijn?
Antwoord : Bij de INPUT-opdracht op regel 10 hebben wij de MSX al verteld hoeveel getallen er komen. Dan hoeft hij ze dus niet meer te tellen. Het aantal staat al vast in het "vakje" met de naam N.

In het tweede voorbeeld berekent de MSX voor ons de som van alle oneven getallen van 1 t.e.m. een getal dat we zelf ingeven op regel 20. Tenminste... dat zouden we willen. Maar in het voorbeeld zitten vier fouten. Zoek die op en verbeter ze:

```
10 CLS
20 INPUT "som van oneven getallen van 1 tot en met";X
30 FOR Z=1 TO X
40 S=S+Z
50 NEXT Z
60 PRINT "de som is" Z
70 GOTO 10
```

Aanwijzing: de fouten zitten op regel 30, 60, en 70. Begint S op nul? Welke stap kiezen we? Wordt de som op het scherm gezet?

59. Een variabele stap

In het begin van dit hoofdstuk hebben we gezien dat we begin- en eindwaarde in de FOR-regel met een variabele kunnen aangeven. Wat we nog niet gedaan hebben, is ook de STAP aangeven met een variabele in plaats van een vast getal. Dat gebeurt in het volgende voorbeeld:

```
10 INPUT "begin"; B
20 INPUT "eind"; E
30 INPUT "stap"; S
40 FOR Z=B TO E STEP S
50 PRINT Z
60 NEXT Z
70 PRINT "klaar bij" Z
80 GOTO 10
```

Voer dit programma een paar keer uit met steeds andere waarden voor B, E en S. Wat de MSX precies doet hangt wel van de drie getallen die ingevoerd worden af. Probeer alle mogelijkheden maar eens uit, ook met negatieve en gebroken getallen.

Het volgende voorbeeld laat zien dat in de FOR-regel ook berekeningen mogen staan. Beginwaarde, eindwaarde en stap mogen we in de vorm van een rekensom in de FOR-opdracht zetten. Ook hier hangt het van de grootte van de ingevoerde getallen A en B af wat het programma precies doet.

```
10 INPUT "A=";A
20 INPUT "B=";B
30 FOR Z=(A+B)/2 TO A+B STEP (A-B)/2
40 PRINT Z
50 NEXT Z
60 PRINT "stop bij" Z
```

We hebben al eerder gezien dat de variabelen, die de begin- en de eindwaarde aangeven in de FOR-regel, best in de herhaling mogen veranderen. De eenmaal vastgestelde begin- en eindwaarde in de FOR-regel verandert daardoor niet. We laten de MSX nu duidelijk maken wat er gebeurt als we de lopende variabele in de herhaling veranderen. Dat gebeurt in het volgende programma op regel 30:

```
10 FOR Z=1 TO 10
20 PRINT "REGEL 20: Z="Z;
30 Z=Z+5
40 PRINT "REGEL 40: Z="Z
50 NEXT Z
```

We zien dat het verloop van het programma hierdoor wél beïnvloed wordt. De herhaling wordt in dit geval maar twee keer uitgevoerd i.p.v. de geplande tien keer. Onderzoek nu het effect van:

```
30 Z=1
30 Z=Z-1
30 Z=Z*0.5  (gebruik de punt, niet de komma!)
5  Z=20
```

Conclusie: wees voorzichtig met het veranderen van de lopende variabele!

Een laatste voorbeeld van het gebruik van FOR-NEXT. Hier gebruiken we de FOR-NEXT om een aantal regels op het scherm over te slaan. In dit voorbeeld zijn het er 5.

```
10 CLS
20 PRINT "boven"
30 FOR R=1 TO 5:PRINT:NEXT R
40 PRINT "ik ben nu hier"
50 FOR R=1 TO 300:NEXT R
60 GOTO 30
```

Op regel 30 staat een FOR-NEXT met alleen een PRINT-opdracht ertussen. De drie opdrachten FOR, PRINT en NEXT staan op één regel. Het zijn wel drie verschillende opdrachten, maar dat mag, op één voorwaarde: de opdrachten moeten gescheiden worden met een dubbele punt (:). De MSX heeft er dan geen moeite mee: hij voert de opdrachten op een regel van links naar rechts uit. Per programmaregel mogen we 255 tekens voor opdrachten gebruiken. Helaas geeft de MSX geen waarschuwing als we bij het 255-ste teken zijn. Als we meer op een programmaregel zetten, neemt de MSX toch maar 255 tekens (op 4 tekens na 7 regels op het scherm) op.

Meer opdrachten op één regel heeft voordelen en nadelen:

- vóór : het is zuiniger (daardoor kunnen we meer opdrachten in het geheugen kwijt) en het is soms (zoals hier) overzichtelijker, omdat wat bij elkaar <u>hoort</u> ook op één regel bij elkaar <u>staat</u>;
- tegen: het is vaak onoverzichtelijker en het is moeilijker om syntax errors op te sporen. Als in 30 b.v. een syntax error zou zitten meldt de MSX alleen "Syntax error in 30", maar <u>niet</u> in welke van de drie opdrachten op deze regel!

Op regel 50 staat nog een FOR-NEXT. Deze FOR-NEXT zorgt ervoor dat de MSX even "voor zichzelf" staat te tellen voordat hij weer teruggaat naar regel 30. Ook hier staan twee opdrachten op één regel. Daarom <u>moet</u> er een dubbele punt tussen staan.

Opmerking: Op regel 30 en 50 wordt dezelfde letter R gebruikt in twee verschillende FOR-NEXT. Daar is geen bezwaar tegen.

60. Netjes op het scherm met TAB

Als we de MSX getallen en teksten netjes op het scherm willen laten zetten moeten we daarvoor wat moeite doen. In de vorige hoofdstukken hebben we al gewerkt met komma's en puntkomma's in PRINT-opdrachten. We moesten ook goed op de spaties letten. In deze en de volgende paragrafen gaan we een nieuwe mogelijkheid gebruiken om teksten en getallen goed op het scherm te krijgen. We doen dat met de opdracht TAB (van TABuleer).
De MSX zal laten zien wat deze opdracht doet in het volgende programma:

```
10 CLS
20 PRINT "0123456789"
30 PRINT "A"
40 PRINT TAB(5)"A"
50 PRINT TAB(10)"A"
```



Het scherm van de MSX is verdeeld in 24 regels van elk 37 posities. De posities op een regel zijn genummerd van 0 t.e.m. 36. Helemaal links zit positie 0, helemaal rechts positie 36.
Met de TAB kunnen we de MSX vertellen op welke positie een tekst of een getal neergezet moet worden. Die positie schrijven we tussen de haakjes. De TAB kan alleen in PRINT-opdrachten gebruikt worden. In dit voorbeeld komt de eerste letter A op positie 0. Er staat geen TAB in de PRINT opdracht op regel 30. Op regel 40 zegt de TAB dat de volgende A terecht moet komen op positie 5. Op regel 50 staat positie 10 tussen de haakjes in de TAB functie. Met behulp van de TAB functie kunnen we o.a. kolommen maken.

```
10 CLS
20 PRINT "ONEVEN" TAB(15) "EVEN"
30 FOR A=1 TO 20 STEP 2
40 PRINT A TAB(15) A+1
50 NEXT A
```

In het volgende voorbeeld maakt de MSX drie kolommen op het scherm.
Het voorbeeld laat op regel 30 zien dat we in een FOR-regel ook getallen met cijfers achter de decimale punt mogen gebruiken.

```
10 CLS
20 PRINT "A" TAB(15) "A*2" TAB(30) "A*3"
30 FOR A=0.1 TO 1 STEP 0.1
40 PRINT A TAB(15) A*2 TAB(30) A*3
50 NEXT A
```

Elke regel op het scherm heeft 37 posities, genummerd van 0 t.e.m. 36. Het volgende programmaatje laat dat zien:

```
10 CLS
20 FOR J=1 TO 4
30 PRINT "0123456789";
40 NEXT J
```

We moeten er bij het gebruik van de TAB dus rekening mee houden dat we met positie 0 beginnen en dat 36 de laatste positie op de regel is. Hoe dat precies uitpakt kunnen we zien in het volgende programma:

```
10 CLS
20 INPUT T
30 PRINT TAB(T) "HIER!"
40 GOTO 20
```

Bij de INPUT-opdracht op regel 20 kunnen we zelf een positie opgeven. Probeer het programma uit met voor T bijvoorbeeld 0, 37, 100, en -3. Probeer ook 35. De positie bestaat, maar op deze regel kan niet meer de hele tekst staan. Wat doet de MSX? Probeer daarna 200, 255 en 256. TAB (256) blijkt "illegaal".

Tussen de haakjes in de TAB-functie mag ook een variabele staan. In het laatste voorbeeld werd de waarde van deze variabele getypt op het toetsenbord op regel 20 bij de INPUT-opdracht. In het volgende voorbeeld maken we van het getal tussen haakjes in de TAB-functie een lopende variabele in een FOR-NEXT:

```
10 CLS
20 FOR T=0 TO 10 STEP 2
30 PRINT TAB(T) "*"
40 NEXT T
```

De bedoeling is dat T loopt van 0 tot 10 met stapjes van twee. Dus: 2 4 6 8 10. Deze waarden krijgt T in de TAB-functie op regel 30. De sterretjes komen dus steeds twee posities uit elkaar te staan en telkens op een volgende regel.
Probeer dit programma ook uit met de volgende opdrachten op regel 20:

```
20 FOR T=10 TO 0 STEP -2
20 FOR T=0 TO 36
20 FOR T=36 TO 0 STEP -1
20 FOR T=20 TO 50
```

61. Niet alles kan met TAB

Typ het volgende programma nauwkeurig over:

```
10 CLS
20 FOR J=0 TO 35: PRINT TAB(J) "*";:NEXT
30 PRINT
40 FOR J=0 TO 35 STEP 2: PRINT TAB(J) "*";:NEXT
50 PRINT
60 FOR J=10 TO 20:PRINT TAB(J)"*";:NEXT
70 PRINT
80 FOR J=20 TO 10 STEP -1: PRINT TAB(J)"*";:NEXT
90 PRINT
100 FOR J=36 TO 0 STEP -1: PRINT TAB(J)"*";:NEXT
```

Controleer het programma voordat het uitgevoerd wordt: Staan alle dubbele punten op de juiste plaats? Zijn de puntkomma's niet vergeten?

Als we dit programma uitvoeren zien we het volgende:

a. Op regel 20 worden 36 sterretjes naast elkaar op één regel gezet.
b. Op regel 30 staat een PRINT-opdracht die ervoor zorgt dat we naar de volgende regel gaan. Dat is nodig omdat achter de PRINT-opdracht op regel 20 een puntkomma staat.
c. Regel 40 laat zien hoe we bij het zetten van de sterretjes steeds een positie overslaan door STEP 2 te gebruiken.
d. Regel 60 zet 11 sterretjes van positie 10 t/m positie 20.
e. Regel 80 laat zien dat de TAB-functie niet op dezelfde regel terug kan. We kunnen op deze regel niet terug van positie 20 naar 19, 18, 17 enz. De sterretjes komen dus toch achter elkaar vanaf positie 20.
f. Regel 100 laat dat ook zien. Alleen beginnen we daar al op positie 36, dus helemaal rechts op de regel. De 37 sterren kunnen op die regel niet geplaatst worden.

We kunnen dus zeggen: met de TAB-functie kunnen we wel steeds verder naar rechts op dezelfde regel, maar niet terug naar een positie links op de regel.

62. Vragen en opdrachten

We besluiten dit hoofdstuk met vragen en opdrachten.
Hier komen eerst de vragen:

Vraag 1:
Wat moet op de punten ingevuld worden om twintig sterren achter elkaar op het scherm te krijgen:

```
10 FOR P=1 TO ..
20 PRINT "*";
30 NEXT P
```

Vraag 2:
Wat moet op de punten ingevuld worden om tien sterren achter elkaar op het scherm te krijgen?

```
10 FOR Q=10 TO ..
20 PRINT "**";
30 NEXT Q
```

Vraag 3:
Hoeveel sterren komen onder elkaar op het scherm als het volgende stukje programma uitgevoerd wordt?

```
10 CLS
20 FOR R=20 TO 10 STEP -2
30 PRINT TAB(10) "*"
40 NEXT R
```

Opdracht 1:
Wat doet het volgende programma?

```
10 CLS
20 FOR A=1 TO 4
30 FOR B=0 TO 9
40 PRINT TAB(B) B
50 NEXT B
60 NEXT A
```

Opdracht 2:
In opdracht 1 zijn twee FOR-NEXT-herhalingen in elkaar gevlochten. Dit heet "ingebed" of "genest" (Engels: nested, als een nest schalen).
In het volgende experiment zijn drie herhalingen ingebed. Wat doet dit programma?

```
10 CLS
20 FOR J=1 TO 5
30 FOR T=1 TO 4
40 FOR K=15 TO 25
50 PRINT TAB(T+K)"*";
60 NEXT K
70 PRINT
80 NEXT T
90 NEXT J
```

Opdracht 3:
Werkt dit programma goed?

```
10 CLS
20 FOR A=1 TO 10
30 PRINT A "TOT DE DERDE IS" A^3
40 NEXT A
50 PRINT
60 A=20
70 GOTO 30
```

Opdracht 4:
Maak een programma, dat het produkt van een rijtje ingevoerde getallen uitrekent en op het scherm zet. Kijk de kunst af in dit hoofdstuk.

Opdracht 5:
Maak een programma, dat voor alle honderdtallen van 100 tot en met 1000 uitrekent hoeveel de helft, het derde en het vierde deel is. Op het scherm moet dus ongeveer dit komen:

```
GETAL      HELFT      DERDE      VIERDE

100         50        33.3333     25
200        100        enzovoorts.
```

Aanwijzing: We kunnen het beste wat experimenteren met TAB om nette kolommen op het scherm te krijgen. Dat is een kwestie van proberen, kijken en verbeteren!

Opdracht 6:
Maak een programma, dat het volgende doet:
- eerst wordt het scherm schoongemaakt;
- daarna worden vier regels "overgeslagen";
- daarna verschijnen drie regels met van links naar rechts allemaal sterren;
- dan weer vier regels "niets";
- weer drie regels sterren;
  en zo maar door.

HOOFDSTUK 8 : Als, dan en anders...

63. Vergelijken

In dit achtste hoofdstuk gaan we één van de belangrijkste mogelijkheden van de MSX gebruiken: het maken van vergelijkingen. We gebruiken daarvoor de opdrachten IF, THEN en ELSE.

Wat een vergelijking is en hoe die eruit ziet kunnen we het beste met een eerste voorbeeld duidelijk maken. We krijgen dan ook een idee van wat we met een vergelijking kunnen doen. Typ het volgende programma:

```
10 CLS
20 INPUT "GEHEEL GETAL";A
30 B=A/2
40 C=INT(B)          (INT is van integer)
50 PRINT A "IS ";
60 IF B=C THEN PRINT "EVEN" ELSE PRINT "ONEVEN"
70 GOTO 20
```

Opmerking: de teksten tussen aanhalingstekens mogen ook in kleine letters getypt worden.

We gaan dit programma eerst regel voor regel bekijken voordat we het uitvoeren.

Op regel 10 staat een bekende opdracht. Hiermee maken we het scherm schoon. Regel 20 is ook niets nieuws. We kunnen een getal typen dat opgeborgen wordt in A.
Regel 30 zet in B de uitkomst van de deling: A/2. Ook dat is niets bijzonders.

Op regel 40 staat wel iets nieuws. We gebruiken hier de INT-functie. INT komt van INTeger, dat betekent "geheel getal". Een geheel getal is een getal zonder cijfers achter de komma, of, om het in BASIC te zeggen, zonder cijfers achter de punt.
Regel 40 zegt dat in vakje C de "integer" van vakje B moet komen te staan. Als in vakje B dus een getal met cijfers achter de punt staat, vallen die er af. Straks komen we hierop nog terug.

Op regel 50 staat een PRINT-opdracht die een stukje van een regel afdrukt. De puntkomma zorgt ervoor dat we op dezelfde regel blijven. Op het scherm komt het getal in A met daarachter de tekst IS.
Regel 60 is ook nieuw. Hier staat de vergelijking. Vertaald in gewoon Nederlands staat er: als in B hetzelfde staat als in C, moet de tekst EVEN afgedrukt worden en anders (dus als in B niet hetzelfde staat als in C) moet de tekst ONEVEN afgedrukt worden.

Regel 70 springt terug naar regel 20. Dat is een gewone GOTO-opdracht.

Probeer nu eerst het programma met een paar getallen uit. Daarna gaan we kijken hoe het precies werkt.

We gaan het programma op papier volgen met twee voorbeelden. Dit is het eerste:

10 scherm schoon
20 getal A: 4
30 B wordt 4/2 is 2
40 C wordt deel van B voor de punt, dus ook 2
50 op het scherm: 4 IS, blijf op deze regel
60 is B=C? Dus: is 2=2? Ja, dus op het scherm EVEN
70 terug naar 20

Nu proberen we het met een tweede voorbeeld:

20 getal A: 5
30 B wordt 5/2 dus 2.5
40 C wordt deel van B vóór de punt, dus 2
50 op het scherm: 5 IS, blijf op deze regel
60 is B=C? Dus: 2.5=2? Nee: dus op het scherm ONEVEN
70 terug naar 20

In het eerste voorbeeld gebruiken we de meest uitgebreide vorm van de vergelijking. Die ziet er in gewone woorden zo uit:

als het antwoord op de vergelijking... ja is, doe dan... en doe anders...

We kunnen het ook zo zeggen: is...? Zo ja, dan... Zo nee, dan...

Vergelijken met een computer is altijd een kwestie van welles of nietes, van ja of nee. Er zijn altijd maar twee mogelijkheden. In computertermen zeggen we dat de vergelijking waar of onwaar is. We kunnen de vergelijking dus ook zo formuleren:

```
vergelijk B=C
indien waar  : PRINT "EVEN"
indien onwaar: PRINT "ONEVEN"
```

Welke formulering we voor onszelf kiezen is niet zo belangrijk. Als het maar (voor onszelf) duidelijk is wat de bedoeling is.

Laat het programma in het geheugen staan voor de volgende paragraaf.

64. Andere vormen

We gaan nu verder met een andere, eenvoudiger vorm van vergelijken. Daarbij kijken we meteen naar alle vergelijkingsmogelijkheden. We beginnen met een heel andere opdracht:

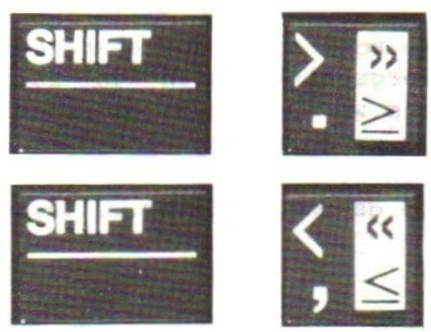
DELETE 30-70

Met de opdracht DELETE 30-70 halen we de regels 30 t/m 70 weg. DELETE betekent verwijderen. Alleen regel 10 en 20 blijven dus in het geheugen staan. Dat is wel zo gemakkelijk omdat in het volgende voorbeeld regel 10 en 20 hetzelfde zijn als in paragraaf 104. De rest van het programma is anders.
In een DELETE opdracht moeten we wel altijd bestaande regels opgeven. Dat kunnen we zien als we bijvoorbeeld nu de volgende opdracht geven:

Wij: **DELETE 100**
MSX: **Illegal function call**

Deze opdracht kan de MSX niet uitvoeren want regel 100 bestaat niet in dit programma. Een "illegale" opdracht dus.

```
10 CLS
20 INPUT "GETAL";A
30 IF A=5 THEN PRINT "PRECIES VIJF"
40 IF A <> 5 THEN PRINT "GEEN VIJF"
50 IF A < 5 THEN PRINT "KLEINER DAN VIJF"
60 IF A > 5 THEN PRINT "GROTER DAN VIJF"
70 PRINT
80 GOTO 20
```



We kijken eerst even naar de vergelijkingen op de regels 30 t/m 60.
Op regel 30 staat: als A gelijk is aan 5, druk dan af "PRECIES VIJF".
Op regel 40 staat: als A niet gelijk is aan 5, druk dan af "GEEN VIJF".
Niet gelijk aan geven we in het programma aan met een "kleiner dan" én "groter dan" teken. Op regel 50 staat: als A kleiner is dan 5, druk dan af "KLEINER DAN 5". Op regel 60 staat: als A groter is dan 5, druk dan af "GROTER DAN 5".
Regel 70 zorgt voor een blanco regel.

Daarmee hebben we nog niet alle mogelijke vergelijkingen gehad, maar probeer eerst dit programma uit. Werkt het? Zo niet, dan zit er een fout in en moeten we het nog eens controleren en verbeteren.
Nu voegen we de laatste twee mogelijkheden aan het programma toe op de regels 35 en 37:

```
35 IF A < =5 THEN PRINT "NIET MEER DAN VIJF"
37 IF A > =5 THEN PRINT "NIET MINDER DAN VIJF"
```

Probeer het programma opnieuw uit met verschillende getallen. We nemen natuurlijk een getal kleiner dan 5, een getal groter dan 5, en precies 5 om alle mogelijkheden te kunnen zien.

Er zijn twee dingen in dit programma waarbij we even stil moeten staan.

a. In dit programma staan alle vergelijkingsmogelijkheden. Hier komen ze nog een keer op een rijtje:

Betekenen:
Is A gelijk aan B? =
Is A niet gelijk aan B? Zijn A en B ongelijk? < >
Is A kleiner dan B? <
Is A gelijk aan B of kleiner dan B? < =
Is A groter dan B? >
Is A gelijk aan B of groter dan B? > =

b. In dit voorbeeld is de vorm van de vergelijking een beetje anders dan in paragraaf 56. Er staat in het Nederlands omgezet niet: **is het antwoord op de vergelijking... ja, doe dan... en doe anders...** want in de opdrachten op regel 30 t/m 60 ontbreekt het laatste stuk: en doe anders...

We moeten ons dus afvragen wat de MSX dan wél doet als het antwoord op de vergelijking geen "ja" is. De oplossing van dit probleem is eenvoudig: als we niet in de opdracht zetten wat de MSX moet doen als het antwoord op de vergelijking niet "ja" is gaat hij altijd gewoon door met de volgende regel. We kunnen dus ook zeggen dat de vergelijking wordt: **is het antwoord op de vergelijking... ja, doe dan... en ga anders naar de volgende regel**

=	<>	>	<	>=	<=
gelijk	niet gelijk	groter	kleiner	groter of gelijk	kleiner of gelijk

In BASIC-opdrachten kunnen we het dus zó zeggen:

```
50 IF A < 5 THEN PRINT "KLEINER DAN VIJF"
```
is hetzelfde als:
```
50 IF A < 5 THEN PRINT "KLEINER DAN VIJF" ELSE GOTO 60
60 IF........ enz.
```

De eerste regel 50 is natuurlijk eenvoudiger.
Als de vergelijking niet waar is, gaan we vanzelf naar de volgende regel.

We gaan de IF-opdracht nu nog eenvoudiger maken. Daarvoor halen we eerst de regels 30 t/m 80 uit het programma dat nog in het geheugen zit:

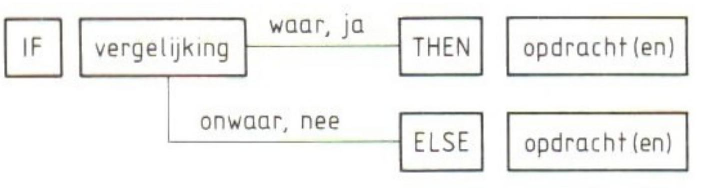
DELETE 30-80

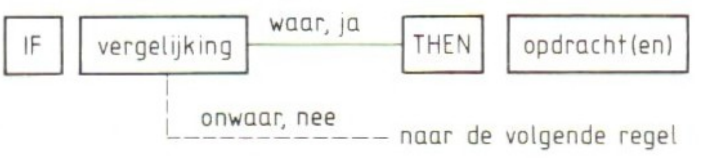
Nu voegen we toe:

30 IF A=5 THEN 20 (denk er bij: ELSE de volgende regel)
40 PRINT "GEEN VIJF"
50 GOTO 20

In regel 30 staat weer een vergelijking. Als A gelijk is aan 5, dan...
Achter deze vergelijking staat alleen nog maar waar de MSX verder moet gaan als de vergelijking waar is (dus als het antwoord op de vraag "Is A=5?" ja is). Op regel 30 staat: als A gelijk is aan 5, ga dan verder op regel 20.
We mogen er zelf bij denken: "en als dat niet zo is, gaan we naar de volgende regel".

Wat doet dit programma als we het uitvoeren? Van elk ingetypt getal controleert hij of het wel of niet gelijk is aan 5.
Als het wel gelijk is aan 5 vraagt hij meteen het volgende getal, is het niet gelijk aan 5, komt op het scherm "GEEN 5" en de MSX vraagt daarna om het volgende getal.





65. Meer mogelijkheden met IF

We kunnen een vergelijking ook zo gebruiken:

als het antwoord op de vergelijking... ja is, sla dan een stuk over en ga verder met regel... en anders door naar de volgende regel.

In het volgende voorbeeld gebruiken we deze mogelijkheid:

```
10 CLS
20 INPUT "GETAL";A
30 IF A>50 THEN 70      (denk er bij: ELSE GOTO 40)
40 A=A+1
50 PRINT A
60 GOTO 30
70 PRINT "WE ZIJN ER!"
80 GOTO 20
```

In dit programma telt de MSX bij het ingevoerde getal net zo lang één op, tot hij bij 50 komt. Als dat tenminste nodig is!

Opmerking: Ook goed zijn de volgende vergelijkingen (de regelnummers zijn slechts voorbeelden):

IF A=B THEN 50 is hetzelfde als
IF A=B THEN GOTO 50 is hetzelfde als
IF A=B GOTO 50

IF A=B THEN 50 ELSE 90 is hetzelfde als
IF A=B THEN 50 ELSE GOTO 90 is hetzelfde als
IF A=B THEN GOTO 50 ELSE GOTO 90 is hetzelfde als
IF A=B GOTO 50 ELSE 90

We hebben nu de drie hoofdvormen van de IF-opdracht gebruikt.
Die drie zien er zo uit:

IF B=C THEN PRINT "EVEN" ELSE PRINT "ONEVEN" (denk er bij: en ga altijd naar de volgende regel).
IF B=C THEN PRINT "EVEN" (denk er bij: zo niet, ga direct naar de volgende regel).
IF B=C THEN 70 (denk er bij: zo niet, ga naar de volgende regel).

De volgende twee programma's doen precies hetzelfde: ze zetten allebei de getallen van 1 t/m 10 op het scherm met daaronder het woord "KLAAR". In het linker programma hebben we de vergelijking met IF gebruikt, in het rechter programma gebruiken we een FOR-NEXT.
Probeer het linker programma uit.

```
10 CLS
20 A=1
30 IF A >10 THEN 70
40 PRINT A
50 A=A+1
60 GOTO 30
70 PRINT "KLAAR"
```

```
10 CLS
20 FOR A=1 TO 10
30 PRINT A
40 NEXT
50 PRINT "KLAAR"
```

Met de vergelijking op regel 30 in dit programma leggen we vast wanneer de MSX moet ophouden met het herhalen van de regels 40, 50 en 60 van het programma. In het rechter programma zit dat in regel 20 waar de FOR-opdracht staat. Natuurlijk is het veel gemakkelijker om in een programma zoals dit de FOR-NEXT te gebruiken in plaats van de IF. Het rechter programma is twee regels korter dan het linker.

Er zijn ook programma's waarin we niet met FOR-NEXT kunnen werken. Het volgende is daarvan een voorbeeld:

```
10 CLS
20 A=2
30 PRINT A
40 A=A*2
50 IF A < 32000 THEN 30      (denk er bij: zo niet, dan naar de volgende regel)
60 PRINT "KLAAR"
```

In dit programma kunnen we de FOR-NEXT niet gebruiken, omdat we van tevoren niet weten hoe vaak de herhaling uitgevoerd moet worden. We kunnen dus niet zeggen: doe dit maar tien of twintig keer. We willen dat de MSX ophoudt als de uitkomst te groot is. Dat hebben we op regel 50 gezet met de vergelijking: zolang A kleiner is dan 32000 gaan we terug om de PRINT-opdracht en de vermenigvuldiging nog eens uit te voeren. Als A niet meer kleiner is dan 32000 gaan we naar de volgende regel en zetten "KLAAR" op het scherm.

In het volgende voorbeeld typen we een rij getallen en gaan van elk getal bekijken of het kleiner is dan 10. Het programma telt hoeveel getallen kleiner zijn dan 10.

```
10 CLS
20 INPUT "HOEVEEL GETALLEN";N
30 FOR I=1 TO N
40 INPUT "GETAL";G
50 IF G < 10 THEN T=T+1
60 NEXT I
70 PRINT T "GETALLEN KLEINER DAN 10"
```

Dit programma hebben we al twee keer eerder gebruikt. Eén keer om een rij getallen op te tellen en een tweede keer in één van de experimenten in het vorige hoofdstuk om het produkt van een rij getallen uit te rekenen.
Nu staat er op regel 50 een vergelijking. Als G kleiner is dan 10 tellen we 1 bij T op en gaan door naar regel 60. Als G niet kleiner is dan 10 slaan we het tellen over en gaan direct naar regel 60.
We kunnen het programma herhalen als we de volgende regels toevoegen:

```
15 T=0
80 GOTO 15
```

Regel 15 zorgt ervoor dat we elke keer met 0 beginnen te tellen. Probeer het programma nu uit met bijvoorbeeld de volgende getallen:

N=6 Getallen: 4 -6 12 10 18 24

N=4 Getallen: 10 10 10 10

N=3 Getallen: 12 8 13

Bij het laatste rijtje getallen ontdekken we een schoonheidsfoutje.
Op het scherm staat de volgende tekst: 1 GETALLEN KLEINER DAN 10. Dat is natuurlijk niet goed. De MSX is niet zo goed in taal. Hij weet zélf niet dat het moet zijn: "1 GETAL" in plaats van "1 GETALLEN". Hij doet alleen precies wat wij in het programma gezet hebben en daar staat dat hij altijd GETALLEN op het scherm moet zetten. Dat is zo op te lossen:

```
10 CLS
15 T=0
20 INPUT "HOEVEEL GETALLEN";N
30 FOR I=1 TO N
40 INPUT "GETAL";G
50 IF G<10 THEN T=T+1
60 NEXT I
70 IF T=1 THEN PRINT T "GETAL"; ELSE PRINT T "GETALLEN";
80 PRINT " KLEINER DAN 10"
90 GOTO 15
```

In dit programma zijn de regels 70, 80 en 90 nieuw. De rest van het programma is gelijk gebleven. Op regel 70 staat: als T gelijk is aan 1 zet dan het woord GETAL op het scherm. Zo niet, zet dan het woord GETALLEN op het scherm. De puntkomma achter de twee PRINT-opdrachten zorgt ervoor dat we op dezelfde regel op het scherm blijven.

De tekst van regel 80 kan er dan keurig achter gezet worden. Hier hebben we dus weer een mogelijkheid om de opdracht IF te gebruiken. We gaan nu op zoek naar een tweede probleem. Dat ontdekken we als we het programma met de volgende getallen uitvoeren:

N=3 Getallen: 12 8 13

N=0 Getallen: 8

N= -5 Getallen: 12

Wat is nu het probleem? Het probleem is dat als we 0 getallen willen typen de MSX tóch om een getal vraagt. Als we typen dat we -5 getallen willen invoeren, wat natuurlijk helemaal niet kan, vraagt de MSX óók om een getal. Eerder hebben we dat probleem ook al gesignaleerd. Nu kunnen we er een oplossing voor bedenken. We kunnen in het programma een regel opnemen waardoor het onmogelijk wordt om met 0 getallen of met bijvoorbeeld -5 getallen te werken. Dat kan met bijvoorbeeld de volgende regel:

25 IF N< =0 THEN PRINT "MAG NIET"

Met deze vergelijking op regel 25 vragen we of het ingevoerde aantal kleiner is dan, of gelijk is, aan 0. Als dat zo is zetten we de tekst MAG NIET op het scherm. Maar de opdracht is daarmee niet af. Als het antwoord ja is moet er niet alleen op het scherm komen te staan "MAG NIET" maar moeten we ook niet verder gaan met het uitvoeren van het tellen.

25 IF N< =0 THEN PRINT "MAG NIET":GOTO 20

In deze nieuwe regel 25 staan achter THEN twee opdrachten, een opdracht PRINT en een opdracht GOTO. Die twee opdrachten worden door de dubbele punt uit elkaar gehouden.

Het volgende is nu erg belangrijk: **de twee opdrachten achter THEN, de PRINT en de GOTO worden allebei alleen uitgevoerd als het antwoord op de vergelijking "is N kleiner dan of gelijk aan 0" ja is.**
Als het antwoord op die vergelijking nee is, voeren we de PRINT-opdracht niet uit en ook de GOTO-opdracht niet. We gaan dan direct door naar de volgende regel.

Het programma moet nu helemaal goed werken. Probeer het uit met verschillende aantallen en getallen.

66. Vraag niet wat je al weet

Maak het volgende programma af en probeer het uit met de MSX.

```
10 CLS
20 INPUT "GETAL";B
30 IF B___0 THEN ______
40 PRINT B "IS NEGATIEF"
50 GOTO 20
66 IF B___0 THEN ______
70 PRINT B "IS NUL"
80 GOTO 20
90 PRINT B "IS POSITIEF"
100 GOTO 20
```

Waarom staat op regel 90 geen vergelijking, bijvoorbeeld in de vorm:

```
90 IF B>0 THEN PRINT B "IS POSITIEF"
```

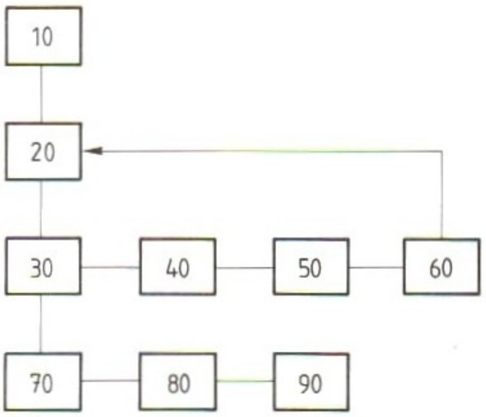
Het antwoord op deze vraag is eenvoudig: op regel 90 is geen vergelijking meer nodig. Dat zit zo: op regel 30 hebben we uitgezocht of het getal B negatief is of niet. Alle negatieve getallen worden afgeleid via regel 40 en 50. Met de getallen die niet negatief zijn gaan we verder naar regel 60. Als we op regel 60 komen hebben we dus alleen nog getallen over die 0 zijn of positief.

Op regel 60 leiden we getallen die 0 zijn af langs regel 70 en 80. Met de getallen die positief zijn gaan we naar regel 90. Daar hoeven we dus niet nog eens te vragen of B positief is. Dat kan niet anders, want alleen met de positieve getallen komen we op regel 90 aan.

We kunnen zeggen dat we onderweg steeds getallen laten afvallen. Eerst op regel 30 de negatieve getallen, dan op regel 60 de nul, zodat op regel 90 alleen de positieve overblijven.

We hebben al in een paar programma's een rijtje getallen ingegeven.
Om daarmee te kunnen werken moesten we steeds van tevoren vertellen hoeveel getallen we wilden invoeren. Met een vergelijking kunnen we dat ook anders oplossen. We hoeven dan niet meer de getallen van tevoren te tellen. Zet het volgende programma in de MSX:

```
10 CLS
20 INPUT A
30 IF A<0 THEN 70
40 B=B+1
50 S=S+A
60 GOTO 20
70 PRINT B "GETALLEN"
80 PRINT "DE SOM IS" S
90 PRINT "GEMIDDELDE IS" S/B
```



In dit programma worden drie dingen gedaan. Op regel 40 wordt geteld hoeveel getallen er ingevoerd worden, op regel 50 wordt de som van de getallen uitgerekend. Op regel 90 wordt het gemiddelde van de getallen uitgerekend en op het scherm gezet. In dit programma vertellen we de MSX niet van tevoren hoeveel getallen we gaan typen. We beginnen gewoon op regel 20 met het eerste getal. Op regel 30 vergelijkt de MSX het getal dat wij in A gezet hebben met 0. Als dat getal groter is dan 0 of gelijk is aan 0 gaat hij verder met de volgende regel 40, 50 en 60. Daarna gaan we terug naar regel 20 voor het tweede getal. Op regel 30 wordt de vergelijking weer gemaakt.

Als wij een getal invoeren dat kleiner is dan 0 (dus negatief) dan gaat de MSX verder met regel 70. Hij zet dan op het scherm hoeveel getallen we getypt hebben, wat de som is en wat het gemiddelde is.

We hebben in dit programma een nieuwe manier bedacht om de MSX te vertellen dat er geen getallen meer komen. Dat doen we door bij de INPUT op regel 20 een negatief getal te typen. Zodra we dat doen is het antwoord op de vergelijking op regel 30 "ja" en gaan we naar 70 en zetten de uitkomsten op het scherm.

We kunnen ook zeggen dat we het rijtje getallen dat geteld en opgeteld moet worden afsluiten met een negatief getal dat niet bij het rijtje hoort.
Deze manier heeft natuurlijk ook een nadeel: met dit programma kunnen we geen negatieve getallen optellen. Zodra we een negatief getal typen "denkt" de MSX dat hij moet ophouden en de uitkomst op het scherm moet zetten.
Als we zo'n programma willen maken moeten we dus een getal zien te vinden dat zelf in het rijtje niet voorkomt. Dat getal kunnen we dan gebruiken om de MSX te vertellen dat we klaar zijn met het typen van de rij getallen. Dat kan bijvoorbeeld zoals hier een negatief getal zijn. We kunnen soms ook het getal 0 gebruiken. De vergelijking op regel 30 wordt dan:

30 IF A=0 THEN 70

Met deze vergelijking kunnen we positieve en negatieve getallen optellen, maar het getal 0 kan niet meedoen. Dat geeft aan dat we klaar zijn.
Soms kunnen we het probleem oplossen met een groot getal of een klein getal, bijvoorbeeld:

30 IF A=9999 THEN 70
30 IF A=-9999 THEN 70

Wat we precies kiezen is niet zo belangrijk. Belangrijk is dat het een getal moet zijn dat zelf in het rijtje op te tellen getallen niet kán voorkomen.
Het is natuurlijk wel belangrijk dat we bij het werken met het programma weten hoe we de rij getallen kunnen afsluiten. We kunnen dat bijvoorbeeld in de INPUT zetten op de volgende manier:

```
20 INPUT "GETAL OF 9999";A
```

We kunnen dan bij het typen van het getal zien dat we óf een "gewoon" getal in moeten typen óf 9999 om de rij af te sluiten.

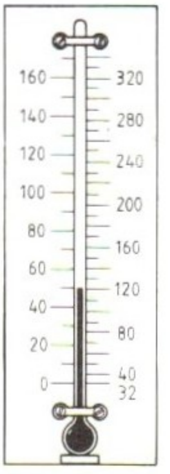
In het volgende voorbeeld staan twee vergelijkingen. De eerste vergelijking op regel 30 gebruiken we om uit te zoeken of we klaar zijn met het typen van het rijtje getallen. We hebben hier gekozen voor een groot getal. De tweede vergelijking op regel 40 heeft een ander doel. Met deze vergelijking gaan we na of het ingevoerde getal wel goed is.
Hier lossen we een probleem op dat we ook al eerder tegengekomen zijn: het delen door nul. Zet het programma in de MSX en kijk of het goed werkt:

```
10 CLS
20 INPUT "GETAL OF 9999";A
30 IF A=9999 THEN 70
40 IF A=0 THEN PRINT "DELEN DOOR 0 MAG NIET" : GOTO 20
50 PRINT "A="A" 1/A=" 1/A
60 GOTO 20
70 PRINT "KLAAR"
```

67. Opdrachten

In deze paragraaf stellen we een aantal vragen over het volgende programma:

```
10 CLS
20 A$="WARM": B$="LEKKER":C$="FRIS":D$="KOUD"
30 INPUT "TEMPERATUUR";T
40 IF T>40 THEN 170
50 IF T<-30 THEN 170
60 IF T>8  THEN 90
70 PRINT D$
80 GOTO 30
90 IF T>14 THEN 120
100 PRINT C$
110 GOTO 30
120 IF T>22 THEN 150
130 PRINT B$
140 GOTO 30
150 PRINT A$
160 GOTO 30
170 PRINT "EINDE"
```



Probeer eerst de vragen te beantwoorden zonder het programma uit te voeren.

Vraag : Wat komt op het scherm als we achter "TEMPERATUUR?" voor T typen:

a. 16 ....

b. 18 ....

c. -4 ....

d. 0 ....

e. 34 ....

f. 8 ....

g. 14 ....

h. 60 ....

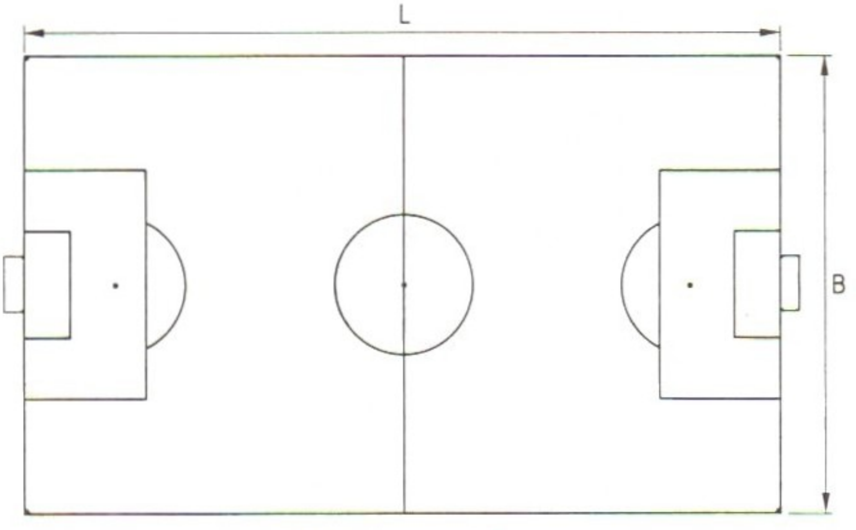
i. 22 ....

j. -50 ....

k. 10 ....

Opdracht 1:
Maak een programma, dat inches omrekent in centimeters.
Het aantal inches wordt getypt, de uitkomst komt op het scherm. Het programma stopt als we 0 typen.

Opdracht 2:
Maak een programma, dat de omtrek en de oppervlakte van een rechthoek uitrekent. Lengte en breedte moeten we kunnen typen.
Negatieve getallen mogen niet en we moeten het programma kunnen stoppen.



oppervlak = L * B

Opdracht 3:
Maak een programma, dat het produkt, de som en het gemiddelde van een rij getallen uitrekent. De getallen worden getypt.
Ze mogen negatief zijn, maar niet nul. Zorg ervoor, dat de MSX niet hoeft te delen door nul.

HOOFDSTUK 9: Functies, kleuren en cassette

68. Opdrachten en functies

In dit hoofdstuk houden we ons o.a. bezig met de volgende BASIC-functies:

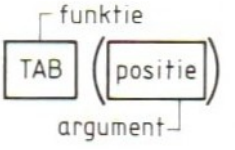
TAB (van TABulator)
INT (van INTeger)
RND (van RaNDomize)

Twee van deze functies, de TAB en de INT, zijn we al een keer tegengekomen.

In de programmeertaal BASIC kennen we opdrachten en functies. Opdrachten in BASIC zijn bijvoorbeeld de bekende: PRINT, IF ... THEN en GOTO. Met deze opdrachten vertellen we de MSX dat hij iets moet doen. De functies in BASIC staan altijd in een opdracht. We hebben bijvoorbeeld al gezien dat we de functie TAB alleen maar kunnen gebruiken in een PRINT-opdracht. Fout is dus:

```
10 TAB(10)
20 PRINT "hier is positie 10"
```

Een functie bestaat altijd uit een woord en iets tussen haakjes.
Tussen de haakjes kan staan
a) een constante,
b) een variabele,
c) een berekening.



Voor de TAB-functie kan dat bijvoorbeeld zijn:
TAB(getal) b.v. TAB(10)
TAB(variabele) b.v. TAB(A)
TAB(berekening) b.v. TAB(A+3)

In de meeste functies gaat het om een getal. De variabele moet dan een getalvariabele zijn en de berekening moet een getal opleveren. Het getal dat tussen haakjes staat heeft in de functie een bepaalde betekenis. In de TAB-functie geeft dit getal aan op welke positie op de regel een getal of tekst neergezet moet worden. Wat in de functies tussen haakjes staat noemen we een argument van de functie. Alle functies zien er dus zo uit (we hebben weer de TAB als voorbeeld genomen): TAB(argument).

Voor het argument van een functie gelden regels, die per functie verschillen. Kijken we weer naar de functie TAB, dan moet het argument een positie op de regel aangeven. De laagste positie is nul, de hoogste 36.

69. De functie INT

De functie INT ziet er zo uit:

INT(argument)   (INT van INTeger)

De functie INT zegt dat van het argument (de constante die tussen de haakjes staat, de getalvariabele of de uitkomst van de berekening) het "naast lagere" gehele getal genomen moeten worden. De functie INT moet dan in een opdracht staan. Bijvoorbeeld zo:

```
10 PRINT INT(2.75)
20 A=INT(3.002)
30 PRINT A
40 IF A=INT(A) THEN PRINT "A IS EEN GEHEEL GETAL"
50 B=INT(0.01) + INT(1.99)
60 PRINT B
```

De functie INT kan in alle opdrachten en functies voorkomen waarin een getal mag staan.

Wat INT precies doet laat het volgende voorbeeld zien:

Wij: **PRINT INT(2.5)**
MSX: **2**
Wij: **PRINT INT(-2.5)**
MSX: **-3**
Wij: **PRINT INT(3.37), INT(-3.37)**
MSX: **3   -4**

Voorbeelden (controleer met de MSX of ze kloppen!):

A	INT(A)
10	10
10.2	10
10.9	10
10.9999	10
-10	-10
-10.2	-11
-10.9999	-11
-0.999999	-1
-0.9	-1

Bij de positieve getallen mogen we zeggen dat de MSX de cijfers achter de punt weglaat. Bij de negatieve getallen geldt dat niet. Er zijn twee functies die op de INT lijken. In de experimenten zullen we daarmee kennismaken.

Opmerking: de MSX werkt met getallen van 14 cijfers. Als we de INTeger nemen van een getal met meer cijfers, rondt de MSX eerst op het 15e cijfer af. Probeer bijvoorbeeld INT(10.9999999999999)

De INT-functie hebben we eerder gebruikt om te bepalen of een getal wel of geen cijfers achter de komma heeft. We deden dat om te kunnen zien of een getal deelbaar was door twee (even of oneven). Hier komt dat programma nog een keer in een iets andere vorm:

```
10 CLS
20 INPUT "GETAL";G
30 IF G=0 THEN STOP
40 INPUT "DEELBAAR DOOR";D
50 IF D=0 THEN PRINT "DELEN DOOR 0 MAG NIET":GOTO 20
60 A=G/D
70 PRINT "GETAL" G "IS ";
80 IF A=INT(A) THEN PRINT "WEL"; ELSE PRINT "NIET";
90 PRINT " DEELBAAR DOOR" D
100 PRINT
110 GOTO 20
```

In regel 30 staat een STOP-opdracht. Als het ingetypte getal G gelijk is aan nul, stopt het programma.

Op regel 50 kijken we of voor D het getal 0 getypt is. Delen door 0 mag niet. Daarachter staat, achter de dubbele punt: GOTO 20. Dat betekent dat we nu een nieuw getal moeten typen. We hadden hier ook kunnen zetten GOTO 40, dan hadden we alleen een nieuwe D hoeven typen.
Op regel 70 staat het eerste deel van de regel die op het scherm moet komen. De puntkomma zorgt dat we op dezelfde regel blijven.
Op regel 80 vergelijken we A (dat kan dus een getal met cijfers achter de komma zijn) met de INTeger van A. Als die twee gelijk zijn, heeft A geen cijfers achter de komma. Het getal G is dan wel deelbaar door D.
Als A en INTeger A niet gelijk zijn, is het getal G niet deelbaar door D. Eén van de twee wordt op het scherm gezet. De puntkomma zorgt ervoor dat we op dezelfde regel blijven.
In regel 80 staan twee PRINT-opdrachten. Er staat dus ook twee keer een puntkomma. In regel 90 maken we de regel op het scherm af. Regel 100 zorgt dat we een regel overslaan voordat we weer een nieuw getal typen.

Voer het programma nu uit en kijk of het werkt. Let er daarbij op dat er in de regel op het scherm op de juiste plaatsen spaties komen, zodat getallen en teksten niet tegen elkaar aan staan.
Probeer ook uit of het programma goed werkt met negatieve getallen.

70. De functie ABS

Als een programma niet goed werkt met negatieve getallen kunnen we twee dingen doen.

a. We kunnen ervoor zorgen dat het programma negatieve getallen niet accepteert. In het voorbeeld uit paragraaf 124 kan dat bijvoorbeeld met de volgende opdrachten:

```
25 IF G < 0 THEN PRINT "FOUT":GOTO 20
45 IF D < 0 THEN PRINT "FOUT":GOTO 40
```

De tekst die op het scherm komt na het typen van een negatief getal kunnen we natuurlijk zo uitgebreid maken als we willen.

b. We kunnen ervoor zorgen dat het programma ook goed werkt met negatieve getallen. Dat kan wel eens lukken door van negatieve getallen gewoon positieve te maken. We kunnen daarvoor een functie gebruiken die er zo uitziet:

ABS(argument) (ABS van ABSolute waarde)

Voorbeelden (probeer of ze juist zijn met de MSX):

A	ABS(A)
6	6
-6	6
2	2
-2	2
3.74	3.74
-3.74	3.74

De functie ABS geeft de ABSolute waarde van het argument, dus van het getal, variabele of de uitkomst van de berekening tussen de haakjes achter het woord ABS. We kunnen deze functie verwerken in het programma, bijvoorbeeld met de volgende regel:

```
65 A=ABS(A)
```

Met deze opdracht zorgen we ervoor dat het getal in A in elk geval niet negatief is. De getallen G en D zelf laten we zoals ze zijn. In regel 80 vergelijken we nu in elk geval twee positieve getallen met elkaar.
In plaats van regel 65 toe te voegen kunnen we de ABS-functie ook zo in het programma verwerken:

```
60 A=ABS(G/D)
```

Ook de functie ABS kan alleen in een opdracht voorkomen.

71. De functie RND

RND komt van RaNDomize. Met deze functie kunnen we de MSX een willekeurig getal uit laten zoeken. De functie ziet er zo uit:

RND(argument)   (RND van RaNDomize)

Het bijzondere van het argument in deze functie is dat het niet belangrijk is welk getal we tussen haakjes schrijven. Voorlopig hoeven we er alleen maar voor te zorgen dat het een positief getal is. In de voorbeelden in dit boek zullen we het getal 5 nemen, maar een ander getal proberen kan ook geen kwaad. Typ het volgende programma, voer het uit en kijk wat het resultaat is.

```
10 CLS
20 FOR I=1 TO 10
30 PRINT RND(5)
40 NEXT I
```

Dit programma zorgt ervoor dat er tien willekeurige getallen op het scherm komen.
Het zijn allemaal getallen tussen 0 en 1 met 14 cijfers achter de komma. De getallen zijn willekeurig, dat betekent: de MSX heeft ze uitgezocht en we weten van tevoren niet wat er komt. Een soort loterij dus! We zullen nu van deze willekeurige getallen eerst bruikbare getallen gaan maken.
Dat kunnen we doen door het getal dat de MSX uitgezocht heeft te vermenigvuldigen met bijvoorbeeld 100. Daarna laten we de cijfers achter de decimale punt weg met de INT-functie.
Probeer dat met het volgende programma.

```
10 CLS
20 FOR I=1 TO 20
30 A=RND(5)
40 A=A*100
50 A=INT(A)
60 PRINT A
70 NEXT I
```

Welke getallen kunnen we verwachten als we vermenigvuldigen met 100 en er dan een INTeger van maken?

	RND	*100	INT
kleinste	0.0000000000001	0.000000000001	0
grootste	0.9999999999999	99.999999999999	99

Er komen getallen van 0 t/m 99 te voorschijn. Probeer het programma nu uit met andere berekeningen op regel 40.

```
40 A=A*10
40 A=A*11
40 A=A*43
40 A=A*6
```

Welke getallen krijgen we nu? Dat kunnen we het beste aan de MSX vragen met het volgende programma:

```
10 CLS
20 K=0.00000000000001
30 G=0.99999999999999
40 INPUT "VERMENIGVULDIG MET";V
50 IF V=0 THEN STOP
60 PRINT "GROOTSTE IS" INT(G*V)
70 PRINT "KLEINSTE IS" INT(K*V)
80 PRINT
90 GOTO 40
```

Met dit programma vermenigvuldigen we het kleinste en het grootste getal dat de MSX kan bedenken met een getal V dat we op regel 40 typen. Als we 0 typen stopt het programma.
Van het grootste en het kleinste getal, vermenigvuldigd met V, maken we dan een INTeger.
Bekijk het resultaat van dit programma met de getallen in het volgende overzicht. Vul in wat de uitkomst is.

V=	kleinste	grootste
10	......	......
11	......	......
43	......	......
6	......	......

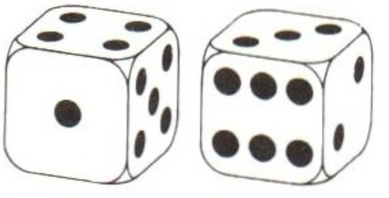
72. Dobbelen

In deze paragraaf maken we van de MSX een dobbelsteenwerpmachine. We laten de MSX een willekeurig getal uitzoeken dat we vermenigvuldigen met 6. Daarna maken we er een INTeger van. Daardoor krijgen we een getal tussen 0 en 5. Op een dobbelsteen staan de getallen 1, 2, 3, 4, 5 of 6. Dat kunnen we krijgen door bij het getal dat de MSX uitgezocht heeft 1 op te tellen.

```
10 CLS
20 INPUT "WERP";K
30 A=RND(5)
40 A=A*6
50 A=INT(A)                    (dat wordt dus 0,1,2,3,4 of 5)
60 A=A+1                       (dus nu: 1,2,3,4,5 of 6)
70 PRINT "IK WERP" A
80 PRINT
90 GOTO 20
```

We kunnen dit programma korter maken door de berekening van A anders op te schrijven. Dat is gedaan in het volgende voorbeeld op regel 30.

```
10 CLS
20 INPUT "WERP";K
30 A=INT(RND(5)*6)+1
40 PRINT "IK WERP"A
50 PRINT
60 GOTO 20
```



We moeten wel even goed kijken hoe deze regel 30 in elkaar zit en wat de MSX in welke volgorde doet:

Opdracht: A=INT(RND(5)*6)+1

eerst:	RND(5)	stel b.v.:	0.57756392935958
dan:	*6	wordt:	3.4653835761575
dan:	INT	wordt:	3
dan:	+1	wordt:	4

We kunnen de voorrangsregels in hoofdstuk 1 nu dus uitbreiden. Die zien er dan zo uit: eerst de functies; dan machtsverheffen; dan vermenigvuldigen en delen; dan optellen en aftrekken, maar... wat tussen haakjes staat moet eerst.

Probeer of het programma werkt.

We gaan nu controleren of de MSX wel "eerlijk" dobbelt en het programma wat "mooier" maken. Als de MSX eerlijk dobbelt moeten alle getallen van de dobbelsteen ongeveer even vaak aan de beurt komen. Tenminste, als we de dobbelsteen vaak genoeg gooien.

In het volgende programma laten we de MSX de dobbelsteen werpen en tellen hoe vaak elk cijfer aan de beurt komt. We zullen de MSX eerst duizend keer laten werpen. Typ het volgende programma in, controleer het en laat het werken.

```
10 CLS
20 FOR W=1 TO 1000
30 A=INT(RND(5)*6)+1
40 IF A=1 THEN T1=T1+1:GOTO 100
50 IF A=2 THEN T2=T2+1:GOTO 100
60 IF A=3 THEN T3=T3+1:GOTO 100
60 IF A=4 THEN T4=T4+1:GOTO 100
80 IF A=5 THEN T5=T5+1:GOTO 100
90 T6=T6+1
100 NEXT W
110 PRINT T1;T2;T3;T4;T5;T6
120 STOP
```

Werkt het programma? We moeten wel even geduld hebben omdat de MSX wat tijd nodig heeft om zijn dobbelsteen duizend keer te gooien. We zullen eerst het programma even bekijken. De vergelijkingen staan op regel 40 t/m 80. We gebruiken 6 variabelen waarin we tellen. Als de MSX een 1 gooit tellen we in T1, gooit hij een 2 dan tellen we in T2, enz.
Achter elke vergelijking staat dat de MSX verder moet gaan met regel 100. Daarom hoeven we op regel 90 niet meer te vragen of A zes is. Als de MSX bij regel 90 aankomt kan er op de dobbelsteen niets anders dan 6 staan. Na duizend worpen zet de MSX de zes variabelen op het scherm. Daarna stopt het programma. We kunnen nu kijken of de getallen op het scherm ongeveer even hoog zijn. Er mag best wat verschil in zitten want het is zuiver toeval wat er te voorschijn komt. Maar als de verschillen erg groot zijn lijkt het erop dat de MSX vals speelt.
Met de volgende regels maken we het programma wat mooier.

```
15 T1=0:T2=0:T3=0:T4=0:T5=0:T6=0
16 INPUT "HOE VAAK GOOIEN";X
17 IF X<1 THEN STOP
20 FOR W=1 TO X
120 GOTO 15
```

Met regel 15 zorgen we ervoor dat de tellers op nul komen te staan. Als we het programma maar één keer uitvoeren hoeft dat niet, maar als we het willen herhalen is het wel nodig. We hebben alle opdrachten op één regel gezet. Vergeet de dubbele punten niet. Op regel 16 geven we aan, hoe vaak we de MSX willen laten gooien. Op regel 17 kijken we of we een getal kleiner dan 1 getypt hebben. Een negatief aantal keren gooien kan natuurlijk niet en nul keer gooien heeft ook geen zin. Dit programma stopt als we een getal kleiner dan 1 getypt hebben. In de FOR-opdracht op regel 20 is ook een verandering aangebracht. Tenslotte hebben we de STOP op regel 120 vervangen door een GOTO naar regel 15.

Het programma kan nu herhaald worden. We kunnen elke keer typen hoe vaak we de MSX willen laten gooien. Probeer het programma uit met weinig worpen (bijvoorbeeld 10) en heel veel worpen (bijvoorbeeld 10000). Kijk hoe lang hij daarover doet!

We gaan nu proberen het dobbelsteenprogramma wat mooier te maken. In plaats van cijfers willen we nu de ogen van de dobbelsteen op het scherm zien.

```
10 CLS
20 INPUT "WERP";K$
30 A=INT(RND(5)*6)+1
40 CLS
50 FOR R=1 TO 10:PRINT:NEXT
60 IF A=1 THEN PRINT: PRINT TAB(21)"O":PRINT:GOTO 120
70 IF A=2 THEN PRINT TAB(22) "O":PRINT:PRINT TAB(20) "O":GOTO 120
80 IF A=3 THEN PRINT TAB(22) "O":PRINT TAB(21) "O":PRINT TAB(20) "o":GOTO
120
90 IF A=4 THEN PRINT TAB(20) "O_O":PRINT:PRINT TAB(20) "O_O":GOTO 120
100 IF A=5 THEN PRINT TAB(20) "O_O":PRINT TAB(21) "O":PRINT TAB(20) "O_O":
GOTO 120
110 PRINT TAB(20) "OOO":PRINT:PRINT TAB(20) "OOO"
120 FOR R=1 TO 5:PRINT:NEXT
130 GOTO 20                                        ( _ = één spatie)
```

De eerste bijzonderheid is dat op regel 40 elke keer het scherm wordt schoongemaakt. De oude dobbelsteen verdwijnt en er komt een nieuwe voor in de plaats. Twee keer hebben we FOR en NEXT gebruikt om een aantal regels over te slaan. Dat is op regel 50 (voor de dobbelsteen) en op regel 20 (na de dobbelsteen). Op de regels 60 t/m 110 staan de zes verschillende dobbelstenen. We hebben ze gemaakt met PRINT-opdrachten met de TAB-functie. Alle PRINT-opdrachten voor één dobbelsteen staan op dezelfde regel met dubbele punten ertussen.

We hebben ervoor gezorgd dat de dobbelsteen steeds op dezelfde plaats op het scherm terechtkomt. Dat hebben we zo gedaan: scherm schoon, 10 regels overslaan, 3 regels dobbelsteen op positie 20, 21 en 22, 5 regels overslaan.

De dobbelsteen zelf is opgebouwd met letters O op de volgende manier (de punten komen <u>niet</u> op het scherm).

```
1        2        3        4        5        6
. . .    . . O    . . O    O . O    O . O    O O O
. O .    . . .    . O .    . . .    . O .    . . .
. . .    O . .    O . .    O . O    O . O    O O O
```

De tweede bijzonderheid zit in de INPUT-opdracht op regel 20. Deze opdracht dient niet om iets ingevoerd te krijgen: met de getypte tekst in K$ doen we in het programma niets. We gebruiken de opdracht alleen om te voorkomen dat de MSX als een razende met zijn dobbelstenen begint te gooien: we zouden hem nauwelijks kunnen volgen. Daarom laten we de machine elke keer vragen:

MSX: **werp?**

Wij hoeven dan niets anders te doen dan de RETURN in te drukken om de volgende worp te krijgen. Iets anders typen dan alleen RETURN kan wel, maar heeft geen zin. In ons programma laten we de computer niet reageren op wat er getypt is!

Probeer of het programma werkt. We noemen dat: testen van het programma. Dat testen is een belangrijk karwei, dat zorgvuldig moet gebeuren. In ons programma betekent dat bijvoorbeeld dat we er pas zeker van zijn dat het programma helemaal goed werkt als we alle dobbelstenen (1 tot en met 6) een keer gezien hebben. We weten immers pas zeker dat regel 90 helemaal goed geprogrammeerd is als de MSX toevallig een keer 4 gooit.

Werkt het programma? Laat het dan in het geheugen staan. We hebben het in de volgende paragraaf nog nodig.

73. De cassette en CSAVE

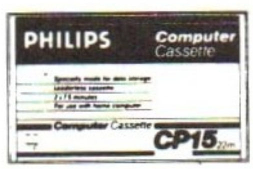
In deze paragraaf gaan we het programma dat we zojuist gemaakt hebben op een cassette zetten. Als een programma op cassette opgeborgen wordt kunnen we het steeds opnieuw gebruiken zonder dat we alle regels van het programma weer opnieuw moeten typen.
We gebruiken daarvoor de opdracht (pas op: nog niet typen!):

CSAVE (C van Cassette, SAVE betekent: bewaren)

Een programma dat op cassette staat kunnen we weer in het geheugen halen met de opdracht (pas op: nog niet typen!):

CLOAD (C van Cassette, LOAD = laden)





BELANGRIJK
Lees eerst het boekje bij de computer:
a. hoe een cassetterecorder aangesloten moet worden;
b. hoe een programma op de cassette geschreven moet worden.

Pas daarna kunnen we het proberen. We gaan ervan uit dat alles goed aangesloten is en dat we weten welke knoppen van de cassetterecorder wanneer ingedrukt moeten worden.
In het volgende nemen we aan dat we werken met een cassetterecorder met afstandsbediening. Gebruik een lege cassette, anders kan het volgende verhaal wel eens niet kloppen.

Opmerking: we kunnen de MSX een cassette niet leeg laten maken. Dat moet met de cassetterecorder zelf gebeuren: sluit niets aan, draai de volumeknop dicht en neem op.

We willen nu het programma dat in het geheugen van de MSX staat overnemen op de cassette.

Wij: **csave "Dobbel"** (nog géén RETURN geven)

Controleer nu of alles goed staat. Zijn de juiste knoppen (ook de knop voor opnemen) ingedrukt?

Wij: **RETURN**
MSX: begint te schrijven
Ok

Zodra Ok verschijnt is de MSX klaar. We kunnen de cassetterecorder stoppen (als de computer dat niet zelf doet via de afstandsbediening).

Opmerking: wij moeten bepalen waar een programma op de band gezet wordt. De MSX kan niet zelf een mooi leeg plaatsje voor ons uitzoeken. Wij moeten dus:
a. zelf de cassette op de juiste plaats instellen (gebruik de tellerstand);
b. zelf de administratie bijhouden van wat er op de cassette staat.

Het is heel belangrijk vanaf het begin goede gewoonten aan te wennen: neem voor elke cassette een stuk papier en noteer wat er opgenomen is met de bijbehorende tellerstanden.

BELANGRIJK
Bij het gebruik van gewone cassettes moeten we eraan denken dat op het aanloopstuk niets geschreven kan worden. Zorg ervoor dat de MSX pas begint te schrijven als het aanloopstuk gepasseerd is.

Op een cassette mogen we zoveel programma's zetten als we maar willen. Om ze uit elkaar te kunnen houden (niet alleen voor ons, maar ook voor de MSX) moeten we elk programma een naam geven. De namen van de programma's op een kant van de cassette moeten verschillend zijn. Anders raken we toch nog in de war.

Een naam op cassette mag hoogstens 6 tekens lang zijn. Minder mag wel, meer niet. Bovendien geldt de regel: het eerste teken van de naam moet een letter zijn.

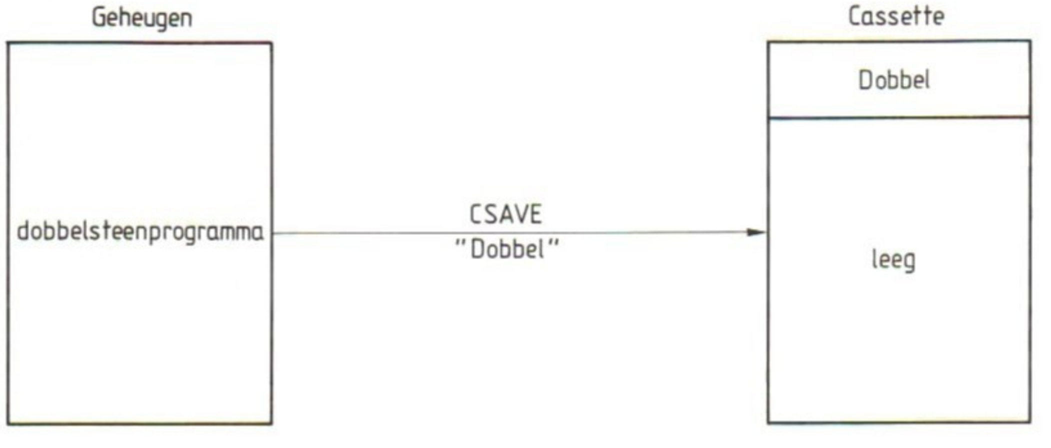
Voorbeelden:

goed	fout
Dobbel	Dobbelsteen *)
dobbel	123
steen1	&aap
steen2	
A-23	

*) Als we toch CSAVE "Dobbelsteen" als opdracht geven, laat de MSX alles wat teveel is gewoon weg. Op de cassette komt dus Dobbel zonder steen!

Staat op het scherm na de CSAVE-opdracht weer Ok? Dan is de MSX inmiddels klaar met schrijven.

We willen nu wel graag zeker weten dat het programma goed op de cassette geschreven is. De situatie is nu zo:



Ons programma staat nu in het geheugen én op de cassette. In het geheugen heeft het programma geen naam. Dat hoeft ook niet, want in het MSX-geheugen kan maar één programma staan. Namen dienen alleen om dingen uit elkaar te kunnen houden en dat hoeft in het geheugen niet. Op de cassette heet het programma Dobbel. Daar heeft het programma wel een naam omdat op de cassette meer programma's kunnen staan.

We willen nu weten of het programma op de cassette hetzelfde is als het programma in het geheugen. Dat kan met een speciale opdracht: de CLOAD?

Spoel eerst de cassette terug naar de plaats waar Dobbel (ongeveer) begint. Stel de cassetterecorder in op "weergeven" (play), maar zorg dat hij nog niet gaat lopen. Als de recorder met de afstandsbediening werkt, zorgt de MSX daar wel voor. Typ dan:

Wij: **cload? "Dobbel"** (nog geen RETURN)
start de cassetterecorder (als de MSX dat niet kan)
RETURN
MSX: **Found:Dobbel**
Ok
Wij: stop de recorder (als de MSX dat niet doet)

We zijn ervan uitgegaan dat het programma goed op de cassette staat. Er kunnen ook andere dingen gebeuren:

a. De MSX meldt: **Verify error**
Het programma op de cassette is niet hetzelfde als het programma in het geheugen. De MSX heeft zijn schrijfwerk op de cassette dus niet goed gedaan en moet het maar over doen: spoel terug naar de juiste plaats en geef een nieuwe CSAVE-opdracht. Controleer daarna weer met CLOAD?

b. De MSX meldt **niets, zelfs geen Found:Dobbel**
De computer is op zoek naar het programma maar vindt niets. Dat kan verschillende oorzaken hebben: staat de volumeknop goed? Stond de cassette wel vóór het programma? Hebben we wel de juiste naam getypt?
Hoe dan ook: de cassetterecorder en de MSX moeten gestopt worden. Stop de MSX met CTRL-STOP. De computer meldt dan: Device I/O error (er is iets mis met een aangesloten apparaat, in dit geval met de cassetterecorder) en Ok. Wij zijn weer aan de beurt. Als de MSX de cassetterecorder niet stil zet, moeten wij dat doen. Spoel terug en probeer opnieuw.

c. De MSX meldt: **Skip :Dobbel**
De computer is gaan lezen en is het programma Dobbel gepasseerd. Hij heeft het niet gelezen, want... Dat kan alleen zijn omdat hij opdracht heeft gekregen iets anders dan Dobbel te lezen, bijvoorbeeld als wij getypt hebben: cload? "dobbel" (met een kleine d). De MSX blijft dan zoeken naar het programma met de verkeerd getypte naam. Omdat hij dat natuurijk nooit vindt, moeten we hem stoppen. Kijk hierboven bij b. hoe dat gaat.

De drie "fouten" die we hier besproken hebben kunnen we gemakkelijk forceren. Het is belangrijk dat we dat ook doen, al kost het even tijd vanwege het spoelen en lezen. We kunnen dan later dergelijke fouten voorkomen of in ieder geval herkennen.

Probeer het zo:

a. spoel de cassetterecorder naar de juiste plaats. Verander dan regel 120 van het programma in het geheugen. Maak ervan:
   120 FOT R=1 TO 6:PRINT:NEXT
   (6 regels overslaan i.p.v. 5)
   Het programma in het geheugen is nu anders dan het programma op cassette!

   Wij: **cload? "Dobbel"**
   MSX: **Found:Dobbel**
   Verify error
   Ok

b. spoel de cassette tot achter het programma Dobbel. Typ dan:

   Wij: **cload? "Dobbel"**
   MSX: leest...
   Wij: **CTRL-STOP**
   MSX: **Device I/O error**
   Ok

c. spoel de cassette tot voor het programma Dobbel. Typ:
   Wij: **cload? "Dobbel"**
   MSX: **Skip :Dobbel**
   leest...
   Wij: **CTRL-STOP**
   MSX: **Device I/O error**
   Ok

Opmerking: Bij het schrijven en later teruglezen van programma's op cassette kan veel mis gaan. Meestal zijn problemen te wijten aan eigen fouten: verkeerde opdrachten, verkeerde aansluiting, verkeerde toetsen op de recorder ingedrukt. We adviseren daarom:

Werk altijd rustig en geconcentreerd en werk direct de administratie bij.

Heel vervelende fouten zijn die waarbij een programma verloren gaat, bijvoorbeeld:

- bij een opdracht CLOAD? de opnameknop van de recorder ingedrukt (cassette wordt gewist);
- in CLOAD? het vraagteken vergeten (programma in het geheugen gewist; zie de volgende paragraaf).

74. CLOAD

Het dobbelsteenprogramma staat nu op cassette onder de naam Dobbel. Met CLOAD? hebben we het gecontroleerd. Heeft de MSX het Oké bevonden? Dan kunnen we verder gaan, anders moeten we het opnieuw opnemen.

We hebben het programma nu op cassette, het "externe geheugen" van de MSX, in veiligheid gebracht. Dat betekent dat we de computer kunnen uitschakelen zonder dat het programma echt verloren gaat.

Wij: **RESET**
MSX: beginbeeld van BASIC
     Ok
Wij: **list**
MSX: Ok
Wij: **run**
MSX: Ok

De computer is heel snel klaar met het uitvoeren van de opdrachten LIST en RUN. Er is immers niets te listen/runnen. Het MSX-geheugen is leeg.

We gaan nu het programma Dobbel lezen en van de cassette overbrengen naar het interne MSX-geheugen. Dat heet: laden van het programma.

Stel de cassetterecorder in. Let op: de opnameknop vooral niet indrukken!

Wij: **cload** (wacht met RETURN)
     druk de afspeelknop (play) in
     RETURN
MSX: **Found:Dobbel**
     Ok
Wij: stop de recorder als de MSX dat niet zelf doet

Reageert de MSX niet met Found:Dobbel en Ok? Dan is er iets mis. Lees gewoon verder.

Op de opdracht CLOAD geven we de MSX opdracht het eerste programma dat hij tegenkomt op de cassette te laden in het interne geheugen. We geven in de opdracht niet op hoe het programma heet. Wij krijgen dan gewoon het eerste programma.

Spoel nu de cassette terug en probeer het opnieuw (denk aan het instellen van de recorder):

Wij: **cload "Dobbel"**
MSX: leest...
     Found:Dobbel
     Ok

In de opdracht CLOAD kunnen we de naam van een programma opnemen. De MSX zoekt dan net zo lang tot hij het programma met die naam gevonden heeft en laadt het vervolgens in zijn geheugen.

De naam die wij opgeven moet natuurlijk wel (letterlijk) juist zijn. Probeer (eerst terugspoelen):

Wij: **cload "dobbel"** (kleine d)
MSX: leest...
Skip :Dobbel
leest...

De MSX blijft voorlopig zoeken, want de cassette is verder leeg. Help de computer met:

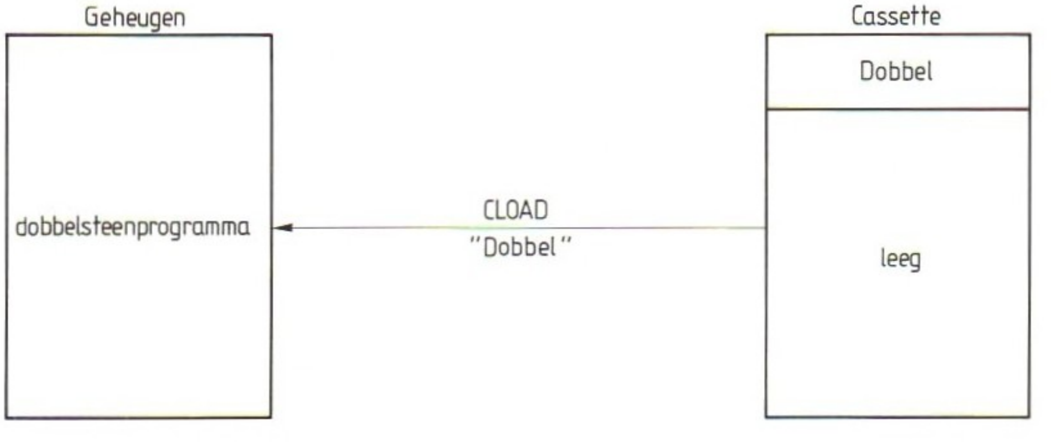
Wij: **CTRL-STOP**
MSX: **Device I/O error**
Ok

Als we een programma in het geheugen willen laden, kunnen we de machine de cassette vanaf het begin laten zoeken. Dat kan echter wel even duren. Het werkt sneller als we zelf de cassette naar (ongeveer) de juiste plaats spoelen en de computer vanaf die plaats laten zoeken en lezen. Maar daarvoor moeten we wel zelf een goede administratie van onze programma's bijhouden!

We gaan nu ons dobbelsteenprogramma serieus laden. RESET de computer, spoel de cassette terug en laad het programma Dobbel:

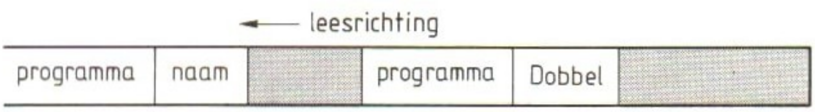
Wij: **clad "Dobbel"**
MSX: **Found:Dobbel**
Ok
Wij: **list** (denk aan het stoppen van de cassetterecorder)
MSX: geeft een overzicht van het programma

Het dobbelsteenprogramma zit dus in het geheugen en kan weer uitgevoerd worden. Probeer het maar. De situatie is nu:



Opmerkingen:

a. Omdat er met een cassette of bij het laden van een programma wel eens iets verkeerd kan gaan is het verstandig er altijd voor te zorgen dat we op een andere cassette een kopie van het programma hebben. We krijgen zo'n kopie door het programma twee keer van het geheugen naar een cassette te brengen. Dat moeten natuurlijk wel twee verschillende cassettes zijn. Als er dan met de ene cassette iets mis gaat hebben we altijd de andere nog. Als er met een cassette of een programma iets mis gaat zorgen we natuurlijk meteen voor een nieuwe kopie.
b. We kunnen de MSX ons laten helpen bij het maken van de administratie van een cassette. Spoel de cassette terug en geef een CLOAD-opdracht voor een programma, waarvan we zeker weten dat het niet op de cassette staat. De MSX gaat dan op zoek en zet bij elk programma dat hij tegenkomt op het scherm:
   Skip :naam van het programma
   Zo krijgen we een mooi lijstje van programma's. Als we dan zelf erbij zetten op welke tellerstanden dat gebeurt, is onze administratie weer compleet.
c. Als we een CLOAD-opdracht geven maakt de MSX het geheugen niet direct leeg. Het oude programma in de machine blijft behouden, totdat de computer echt kan beginnen met laden van een programma. Op dat moment maakt hij het geheugen schoon.
d. De naam van een programma staat aan het begin van het stuk band, waarop het programma staat.



Als de computer dit begin voorbij is herkent hij het programma niet meer. Hij begint dan pas met zoeken en eventueel laden met het eerstvolgende programma op de band. Bij het laden van een programma hoeft de cassette dus niet precies in de lege ruimte tussen twee programma's te staan.

75. Getallen raden

We gaan nu met de MSX een spelletje maken en spelen. Het is een eenvoudig spel waarin we de MSX een getal tussen 1 en 100 laten kiezen, dat wij dan gaan raden. Zet het programma in de MSX en kijk het goed na.

```
10 CLS
20 PRINT "getallen raden"
30 PRINT
40 INPUT "raden (j/n)";A$
50 IF A$="n" THEN STOP
60 A=INT(RND(5)*100)+1
70 INPUT "raad eens";R
80 IF R=A THEN PRINT "GERADEN!":GOTO 30
90 IF R>A THEN PRINT "MINDER" ELSE PRINT "MEER"
100 GOTO 70
```

In de aanwijzing tussen aanhalingstekens in de INPUT-opdracht staat dat we op de vraag "RADEN?" met een letter j of een letter n moeten antwoorden. Let op: J en j zijn voor de MSX niet hetzelfde!
We zien op regel 50 dat we ook letters met elkaar kunnen vergelijken, of beter gezegd: we kunnen ook vergelijkingen maken met strings. We komen daar later op terug.

Vraag : Wat gebeurt er als we bij de INPUT-opdracht op regel 40 bijvoorbeeld een letter P typen?

Speel het spel en kijk of het programma goed werkt.

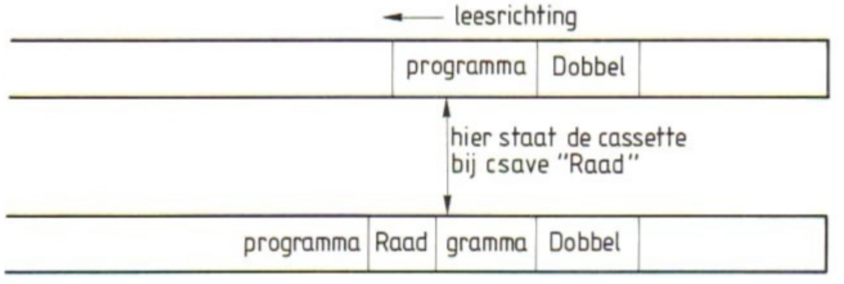
We willen dit programma nu ook op de cassette schrijven (opslaan). De opdracht die we daarvoor gebruiken is bekend. We moeten er echter eerst voor zorgen, dat de cassette op de juiste plaats ingesteld wordt, omdat er anders ongelukken gebeuren (ook daarom is het zo belangrijk altijd voor tenminste één kopie van het programma op een andere cassette te zorgen).

Als we de tellerstanden goed hebben bijgehouden kunnen we de cassetterecorder na het programma "Dobbel" instellen en het nieuwe programma opnemen:

Wij: **csave "Raad"**
MSX: schrijft...
Ok
Wij: terugspoelen tot het begin van Raad
cload? "Raad"
MSX: **Found:Raad**
Ok

Opmerking: als de MSX een "Verify error" geeft moeten we opnieuw beginnen. Zorg ervoor dat de cassette op de juiste plaats staat. Bij het laden kan de MSX zoeken naar het gevraagde programma, bij het schrijven begint hij direct op de plaats waar de cassette staat.

Dat kan het volgende probleem geven:



We zijn niet alleen het programma "Dobbel" kwijt, maar het programma "Raad" werkt ook niet goed meer.

Het programma "Getallen raden" staat nu op de cassette, maar ook nog in het geheugen. Geef een opdracht LIST om dat te zien. We kunnen met het programma gewoon doorwerken. We hebben het alleen op de cassette gezet om ervoor te zorgen dat we het niet kwijt zijn als we een fout maken.

We maken het programma "Getallen raden" wat mooier. We zorgen er eerst voor dat de MSX telt hoeveel beurten wij nodig hebben om een getal te raden. Dat kan met de volgende opdrachten:

```
55 T=0
75 T=T+1
80 IF R=A THEN PRINT "GERADEN IN" T "BEURTEN": GOTO 30
```

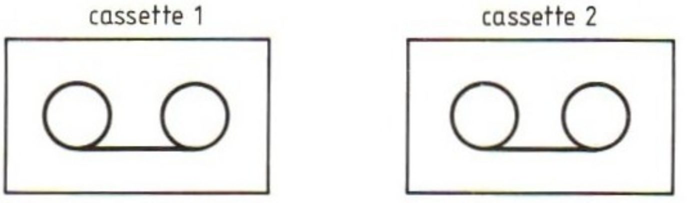
Zet de nieuwe regels 55 en 75 in het programma en verander regel 80. Probeer daarna of het programma goed werkt. Doet het programma het goed? Dan gaan we nu het verbeterde programma op de cassette schrijven. We kunnen dat op twee manieren doen:

a. We zetten het nieuwe programma "Getallen raden" erbij op de cassette. Het oude programma, dat we Raad hebben genoemd, blijft dan staan. Dat gebeurt als we dit nieuwe programma een andere naam geven, bijvoorbeeld Raad-2.
b. We willen het nieuwe programma "Getallen raden" in plaats van het oude programma op de cassette schrijven. Het oude programma mag er dan af. We mogen dit nieuwe programma dezelfde naam geven als het programma dat al op de cassette staat, dus Raad.

In beide gevallen moeten we zelf voor de juiste instelling van de cassette zorgen en dat zo doen dat er niets verloren gaat, dat we willen bewaren.

We hebben intussen al ontdekt dat het helemaal niet zo eenvoudig is. Daarom dit advies:

gebruik tijdens het programmeren altijd twee lege cassettes als "werkcassettes"



We noemen ze cassette-1 en cassette-2

Als we een flink stuk geprogrammeerd hebben schrijven we het programma op cassette-1 én op cassette-2. We hebben dan twee copieën op de band.

Daarna programmeren we verder. Elke keer als we een stuk gedaan hebben leggen we de "volgende versie" uit het geheugen vast op één van de cassettes en wel om en om: versie 1 komt op cassette-1 én 2, versie 2 alleen op cassette-1, versie 3 op cassette-2, versie 4 weer op cassette-1 enzovoorts.

We hebben dan steeds de laatste en de vorige versie in veiligheid gebracht op cassette. We moeten natuurlijk wel even onthouden welke cassette aan de beurt is!

Met dit systeem (dat natuurlijk wel wat tijd kost) kunnen we veilig werken: als we een fout maken of er gaat iets mis, is het verlies altijd beperkt.

76. Spelen met kleur

De MSX kan, als hij aangesloten is op een kleuren-tv, letters, cijfers en andere tekens in kleur op een eveneens gekleurde achtergrond weergeven.

Bij het beginscherm laat de computer witte letters op een donkerblauwe achtergrond zien. Zowel de kleur van de letters als de kleur van de achtergrond kunnen we veranderen met de opdracht COLOR. Het woord COLOR hoeven we zelf niet te typen: we kunnen functietoets F1 of F6 (SHIFT-F1) gebruiken.
Probeer:

Wij: **color 1**
MSX: geeft zwarte letters op een donkerblauwe achtergrond
Wij: **color 15**
MSX: de letters worden weer wit

In deze paragraaf besteden we niet alleen aandacht aan de kleuren van de MSX. Eigenlijk is dat zelfs maar bijzaak. Het gaat om het proberen zelf.

In het BASIC-boekje bij de computer kunnen we vinden hoe de COLOR-opdracht ingevuld kan worden. Er staat zelfs een tabel met kleuren bij.
De opdracht ziet er zo uit:

```
COLOR voorgrondkleur,achtergrondkleur,randkleur
```

Achter het woord COLOR mogen drie kleuren staan, gescheiden door komma's. We hoeven niet altijd drie getallen in te vullen: wat we achterwege laten blijft zoals het was. We gaan nu eerst proberen wat er gebeurt als we tegen deze regel zondigen:

Wij: **color RETURN**
MSX: **Missing operand**

Dit bericht van de computer betekent dat er in de opdracht een noodzakelijk gegeven (de operand) ontbreekt. De computer voert de opdracht niet uit. Geen enkel getal invullen mag dus kennelijk niet.

Wij: **color 1,2,3,4**
MSX: verandert de kleur van het scherm
Syntax error
Ok

De computer begint de opdracht uit te voeren. Hij begrijpt de opdracht tot en met COLOR 1,2,3 en voert dit deel dan ook (van links naar rechts) uit. Wat er dan nog komt begrijpt hij niet meer (,4). Hij meldt dus een syntax error.

Nu willen we graag alle mogelijke combinaties van kleuren uitproberen, maar... dat zijn er nogal wat! Zestien voorgrondkleuren, gecombineerd met 16 achtergrondkleuren, gecombineerd met 16 randkleuren, dat zijn maar liefst 16x16x16 ofwel 4096 mogelijkheden!

Als we alles willen zien moeten we dus 4096 opdrachten geven en dat is toch wel wat veel van het goede.

We roepen daarom de hulp van de MSX zelf in: we maken een programma dat alle mogelijkheden laat zien. We gebruiken daarbij de FOR-NEXT-opdracht, die bij uitstek voor dit soort probeerwerk geschikt is.
Probeer het zo:

```
10 FOR K=0 to 16
20 COLOR K
30 NEXT K
```

Voer dit programma uit:

```
MSX: Illegal function call in 20
     Ok
```

Dit bericht hebben we al eerder gezien. Het betekent: we vragen de MSX iets dat niet mag (dat illegaal is). Wat is illegaal? De opdracht COLOR 16. Het getal dat de kleur aangeeft mag en kan niet kleiner zijn dan 0 (COLOR-1 geeft ook een Illegal function call) en niet groter dan 15.
Dat hebben we dus geprobeerd: als we met het getal na COLOR buiten het bereik van de getallen 0 tot en met 15 komen, krijgen we een "Illegal function call".

Verder hebben we nog niet veel kleur gezien: de MSX verschiet zo snel van kleur, dat wij het niet kunnen volgen. We zullen hem wat langzamer moeten maken.

Dat kan met behulp van een "wachtlus", die we ook al eerder gebruikt hebben. Een wachtlus is een FOR-NEXT-opdracht zonder "echte" opdrachten ertussen. Voeg zo'n wachtlus toe aan het programma!

```
25 FOR W=1 TO 300:NEXT W
```

Voer het programma nu opnieuw uit. Het programma eindigt nog steeds met het bericht over de fout op regel 20.
Verander die:

```
10 FOR K=0 TO 15
```

Opmerking: voor de MSX zit de fout op regel 20, omdat hij daar op een zeker moment de opdracht COLOR 16 moet uitvoeren. De <u>oorzaak</u> van die fout zit echter op regel 10! Die moet dus veranderen.

Gaat het nog te snel? Verander dan regel 25, bijvoorbeeld in:

```
25 FOR W=1 TO 1000:NEXT
```

We kunnen met dit programma de kleurmogelijkheden van de voorgrond zien, maar nog niet erg duidelijk is welke kleur bij welke code hoort. Daarom is het beter iets op het scherm te zetten waaraan we dat kunnen zien, bijvoorbeeld zo:

```
15 CLS
17 PRINT "Kleur nummer" K
```

Voer het programma uit. We zien dan dat er twee kleuren niet op het scherm komen: nummer 0 en nummer 4.
Nummer 0 (transparant) ontbreekt, omdat de letters op het scherm in deze kleur "doorzichtig" zijn. De tekst "Kleur nummer 0" staat wel op het scherm, maar we kijken er dwars doorheen.
Kleur nummer 4 is donkerblauw. De achtergrond is dat ook. De tekst "Kleur nummer 4" staat dus wel op het scherm, maar we zien dat niet omdat het dezelfde kleur is als de achtergrond. Een tekst in schutkleur dus!

Opmerking: tegen de donkerblauwe achtergrond zijn niet alle voorgrondkleuren even goed te gebruiken. Veel hangt ook af van de juiste instelling van het tv-toestel.

We gaan nu ook de achtergrondkleur veranderen. Deze kleur wordt aangegeven met het tweede getal in de COLOR-opdracht. Omdat we alle voorgrondkleuren tegen alle achtergrondkleuren willen zien, doen we het zo:

```
5 FOR A=0 TO 15
7 PRINT "Achtergrondnummer" A
15 gaat eruit  (typ 15 RETURN)
20 COLOR K,A
35 FOR W=1 TO 500:NEXT W
40 NEXT A
```

Breng deze wijzigingen aan en controleer ze. Voer het programma nog niet uit.

We gaan eerst iets doen aan de regelnummers van dit programma. De regels zijn nu zo raar genummerd dat het programma er wat slordig uitziet. Maar daar is een heel eenvoudige oplossing voor:

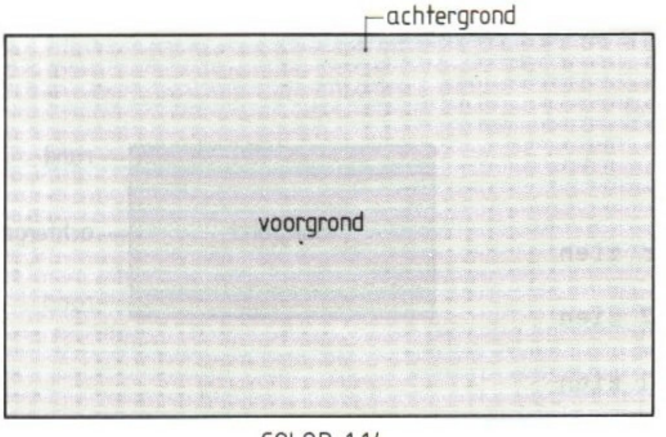
Wij: **RENUM**
MSX: **Ok**

Geef nu maar eens een LIST-opdracht en zie hoe keurig de MSX de regels genummerd heeft.

Voer nu het programma uit en bekijk het resultaat. Dat kost wel even tijd!

Opmerking: als we het programma afbreken met CTRL-STOP kán het zijn dat dat net toevallig gebeurt als de kleur van de voorgrond en de achtergrond gelijk zijn. Als we dan iets gaan typen zien we niets. Geef in dat geval een opdracht COLOR 15,4. We krijgen dan de oorspronkelijke combinatie wit op donkerblauw weer terug.
Bovendien zit er een adderje onder het MSX-gras: het programma eindigt met de kleurcombinatie wit-wit. Als het programma helemaal afgewerkt is, zien we dus ook alleen een geheel wit scherm zonder de (witte) letters van Ok.

Wij: **COLOR 1,14** (we zien het niet tijdens het typen!)
MSX: zwart op grijs
Ok



COLOR 1,14

77. Dobbelen in kleur

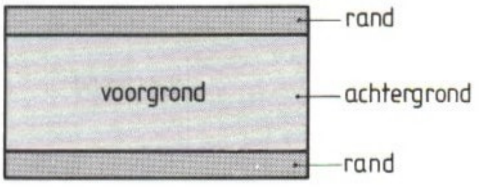
In deze paragraaf gaan we het dobbelsteenprogramma een beetje extra kleur geven, maar daarvoor moeten we eerst nog een andere opdracht bekijken: de SCREEN.

In de vorige paragraaf, over de kleuren van het scherm, was sprake van randkleuren. We hebben daaraan geen aandacht besteed en wel om de volgende eenvoudige reden: het "normale" scherm van BASIC hééft geen randkleuren.

De MSX heeft echter vier soorten schermen, elk aangeduid met een nummer:

0 is het "normale" scherm
1 is het scherm voor randen
2 is voor grafisch werken
3 is voor grafisch werken

We schakelen om van het ene type scherm naar het andere met de SCREEN-opdracht (SCREEN = scherm).
Probeer:



Wij: **SCREEN 1**
MSX: toont schermsoort 1
Wij: **SCREEN 0**
MSX: laat het normale scherm weer zien
Wij: **SCREEN 2**
MSX: laat heel even schermsoort 2 zien
Wij: **SCREEN 3**
MSX: laat heel even schermsoort 3 zien

De schermsoorten 2 en 3 komen later nog even aan de beurt. Wij beperken ons nu tot scherm 0 en 1.

Geef de opdracht:

Wij: **SCREEN 1**
MSX: laat schermsoort 1 zien

Dit scherm heeft een voorgrond (letters), een achtergrond en randen. We kunnen nu de randkleuren even bekijken:

Wij: **COLOR 1,14,13** (een grijs vlak met een paars randje met zwarte letters)

We hoeven kleuren die hetzelfde blijven niet steeds opnieuw in de COLOR-opdracht te zetten. We mogen ze weglaten op één voorwaarde: het moet voor de MSX wel duidelijk zijn wat we bedoelen.

Dat kan bijvoorbeeld zo:

COLOR,,12

zorgt voor een groene rand. In deze opdracht is 12 duidelijk het derde getal, al hebben we de eerste twee (voorgrond en achtergrond) weggelaten. Door de komma's, die de drie getallen uit elkaar houden, wél in de opdracht te zetten kan de computer zich niet vergissen.

COLOR,11

zorgt nu voor een gele achtergrond. De rand blijft groen, de voorgrond blijft zwart. Elf is immers het tweede getal.

COLOR,14,12

zorgt nu weer voor een grijze achtergrond (tweede getal) en groene randen (derde getal).

In de vorige paragraaf hebben we gezien dat de getallen in de COLOR-opdracht ook de vorm van een getal-variabele mogen hebben. Dat is trouwens een typisch kenmerk van BASIC: overal waar een getal mag staan mag ook een getal-variabele of een berekening staan. Er is maar één voorwaarde aan verbonden: de getal-variabele moet een "passende" waarde hebben. Niet te laag, niet te hoog, maar tussen de toegestane grenzen. In het geval van de COLOR-opdracht: een waarde van 0 tot en met 15 en niet daarbuiten. Voor een berekening geldt hetzelfde.

Met deze wetenschap gaan we nu het dobbelsteenprogramma aanpassen. Haal het eerst in het geheugen met de opdracht:

Wij: **CLOAD "Dobbel"**
MSX: **Found:Dobbel**
Ok

Controleer met een opdracht LIST of het programma nog goed is.

Opmerking: we werken nog steeds op schermsoort 1. Het resultaat van een LIST-opdracht ziet er nu iets anders uit dan op het normale scherm! Hoeveel tekens gaan er nu op een regel? Hoeveel regels zijn er? Kunnen we met de cursor ook in de randen komen?

Breng nu de volgende regels in het programma aan:

```
15 SCREEN 1
55 COLOR A+2,A+3,A+4
```

Voer het programma uit en kijk of de kleurcombinaties vervallen. Maak er anders maar iets anders van. We kunnen bijvoorbeeld ook per dobbelsteen de kleur bepalen. Regel 55 moet er dan weer uit en van regel 60 maken we b.v.:

60 IF A=1 THEN COLOR 1,14,13:PRINT TAB(21)"0":GOTO 120

We moeten dan ook regel 70 tot en met 110 voorzien van een passende COLOR-opdracht. Het is ook aardig de letters O, die we gebruikt hebben voor de ogen van de dobbelsteen, te vervangen door een grafisch teken, bijvoorbeeld het gezichtje. Vergeet bij het editten vooral de RETURN op elke veranderde regel niet!

Pas het programma naar eigen wensen aan met COLOR-opdrachten en eventueel andere tekens. Zet het programma daarna weer op cassette onder de naam "Dobbel" met een CSAVE-opdracht. Zorg voor een juiste instelling van de cassetterecorder.

78. Opdrachten

Opdracht 1:
Schrijf de opdrachten van regel 70 tot en met 120 op als één opdracht:

```
60 INPUT B
70 A=RND(4)
80 A=A*100
90 B=ABS(B)
100 A=A*B
110 A=INT(A)
120 A=A+3
```

Opdracht 2:
Schrijf de volgende opdracht op in zoveel mogelijk losse opdrachten:

```
200 A=INT(4*((INT(B/2.5)+INT(C/0.3))/ABS(B-C)))
```

Opmerking: We mogen andere letters erbij gebruiken.

Opdracht 3:
De functies FIX(argument) en CINT(argument) lijken op de INT-functie. Zoek uit wat deze functies doen met het volgende programma:

```
10 CLS
20 INPUT "GETAL";G
30 PRINT "GETAL="G
40 PRINT "INT="INT(G)
50 PRINT "FIX="FIX(G)
60 PRINT "CINT="CINT(G)
70 PRINT
80 GOTO 20
```

Zet de uitkomsten in een tabel met de volgende getallen:

Getal	INT	FIX	CINT
0	...	...	...
0.1	...	...	...
-0.1	...	...	...
0.3	...	...	...
-0.3	...	...	...
0.5	...	...	...
-0.5	...	...	...
0.7	...	...	...
-0.7	...	...	...
0.9	...	...	...
-0.9	...	...	...
1.1	...	...	...
-1.1	...	...	...

HOOFDSTUK 10: Gegevens lezen: READ en DATA

79. DATA en READ

In dit hoofdstuk gaan we gegevens (getallen en teksten) op een nieuwe manier opbergen in het programma. Verder zullen we de MSX regelnummers laten zetten, zodat wij die niet meer hoeven te typen. Onze programma's worden langzamerhand wat groter. We moeten daarom wat duidelijker programmeren, anders raken we zelf de draad gemakkelijk kwijt. We zullen dat doen met toelichting en subroutines.

We hebben al verschillende manieren leren kennen om teksten en getallen in het programma te zetten. Soms kiezen we voor het opbergen van een gegeven (tekst of getal) in een variabele: **T$="CARMEN"** Soms kiezen we voor de tekst of getal in een opdracht: **PRINT "CARMEN"**

De derde manier ziet er zo uit:

```
10 CLS
20 READ A
30 PRINT A
40 GOTO 20
100 DATA 3,10,66,-2,44,3.7,6.6,-0.001,0
```

Voordat we dit programma in de MSX zetten en uitvoeren een paar opmerkingen:

a. Op regel 100 staat een nieuwe opdracht, de DATA. Regel 100 noemen we een dataregel. Eigenlijk staat hier geen opdracht maar geeft DATA aan dat op deze regel gegevens (teksten en getallen) staan. Als de MSX regel 100 tegenkomt voert hij dus geen opdracht uit. Verderop in deze paragraaf zullen we dat zien.
b. Op dataregel 100 staan alleen getallen. De getallen worden uit elkaar gehouden door een komma. Er staan 9 getallen op dataregel 100: het eerste getal is een 3, het laatste een 0.
c. We kunnen met de MSX met getallen en teksten op een dataregel maar één ding laten doen: we kunnen hem opdracht geven een getal of tekst te lezen. Dat doen we met de opdracht READ die op regel 20 staat. In de opdracht READ staat altijd een variabele. De opdracht betekent dat de MSX één getal of tekst van de dataregel naar de variabele A moet brengen.
d. De MSX houdt bij, bij welk getal of welke tekst hij gebleven is. Bij het lezen van gegevens van een dataregel neemt hij steeds de eerstvolgende.

Zet nu dit programma in de MSX. Kijk goed na of de dataregel helemaal correct in het geheugen staat. Zorg voor de komma's op de juiste plaatsen.
Voer het programma dan uit en kijk wat er gebeurt.

De MSX stopt met het volgende bericht:

MSX : Out of DATA in 20

Op regel 10 maakt de MSX het scherm schoon. Op regel 20 moet hij een getal lezen uit een dataregel. De MSX zoekt die dataregel op en vindt hem op regel 100. Uit de dataregel neemt hij het eerste getal, dus de 3, en brengt die naar variabele A. Daarmee is de opdracht op regel 20 uitgevoerd.
Op regel 30 zet de MSX het getal in A op het scherm. Daardoor verschijnt de 3. Op regel 40 moet hij terug naar regel 20. Daar voert hij opnieuw de opdracht READ uit. Hij neemt nu het tweede getal uit de dataregel, dat is de 10, en brengt die naar A. De 3 verdwijnt (wordt overschreven), de 10 komt ervoor in de plaats. Op regel 30 wordt het getal 10 op het scherm gezet. Op regel 40 gaat de MSX weer terug naar regel 20, neemt het derde getal uit de dataregel en zet dat op het scherm. Zo werkt hij door totdat de 0 (het laatste getal uit de dataregel) op het scherm staat. De MSX moet nu opnieuw terug naar regel 20. Daar krijgt hij de opdracht om van de dataregel nog een getal te halen, maar dat getal is er niet meer.
Nu stopt de MSX en zet het bericht op het scherm. Dit bericht betekent: ik kan niet verder, ik moet "lezen", maar er zijn geen gegevens meer.

We kunnen dit zo in schema zetten:

Ronde	A wordt	op het scherm
1	3	3
2	10	10
3	66	66
4	-2	-2
enzovoorts, tot...		
8	-0.001	-1E-03
9	0	0
10	getallen op	Out of DATA

Op dataregels kunnen we ook teksten opbergen. Die teksten hoeven niet tussen aanhalingstekens te staan, behalve als in de tekst een komma (,) of dubbele punt (:) zit. De komma's houden de teksten uit elkaar. Het programma kan er nu zo uitzien:

```
10 CLS
20 READ A$
30 PRINT A$
40 GOTO 20
100 DATA Jan,Henk,Greet,Truus,Jojo
```

Ook hier houdt het programma op als er geen gegevens meer te lezen zijn.
We krijgen opnieuw het bericht: **Out of DATA in 20.**

Moeilijk is dit niet, maar we moeten er wel voor zorgen dat de gegevens in de dataregel netjes opgeborgen worden. Doen we dat niet, dan komt de MSX in de war. Dat kunnen we zien met het volgende programma:

```
10 CLS
20 READ A
30 PRINT A
40 GOTO 20
50 DATA,_4,7,-6;8_3,42,Jan,7,          (_ = spatie)
```

Voer dit programma uit en kijk hoe de MSX reageert. Elke keer als hij een fout signaleert verbeteren we die en proberen het opnieuw, net zo lang tot alle fouten uit de dataregel verdwenen zijn.
We zien dat het noodzakelijk is om de komma's op de juiste plaats te zetten. Met de eerste en de laatste komma heeft de MSX geen moeite: hij neemt gewoon aan dat de DATA-regel begint en eindigt met een getal 0 (vóór de eerste komma en ná de laatste). De puntkomma tussen de -6 en 8 brengt de MSX helemaal in de war, maar de spatie tussen 8 en 3 weer niet! Waar we ook aan moeten denken is dat we teksten niet naar een getalvariabele kunnen brengen. Dat blijkt als de MSX de naam Jan moet lezen in variabele A. We krijgen dan weer het bekende bericht: syntax error 50.

In het programma mogen meer dataregels staan. De plaats van de dataregels is niet belangrijk. De MSX zorgt er zelf voor dat hij de tel goed bijhoudt. Dat kunnen we zien als we het volgende programma in het geheugen zetten en uitvoeren. Denk aan de plaats van de komma's.

```
10 DATA 4,5,6
20 READ A
30 DATA 7,8,9
40 PRINT A
50 DATA 10,11,12
60 GOTO 20
70 DATA 13,14
```

Opmerking: De dataregels staan net als gewone programmaregels in het geheugen. We kunnen ze dus de editten om fouten in een dataregel te verbeteren.

80. Out of DATA

We gaan nu eerst een schoonheidsfoutje wegwerken. Typ het volgende programma en voer het uit:

```
10 CLS
20 READ A
30 PRINT A
40 GOTO 20
50 DATA 3,10,7,12,44,63
```

De MSX stopt vanzelf met het programma als de gegevens op zijn. Het bericht "Out of DATA" verschijnt op het scherm. Het programma wordt dan afgebroken. Daar zitten twee nadelen aan: in de eerste plaats staat het natuurlijk niet zo mooi als het programma eindigt met de melding van een fout. In de tweede plaats - en dat is belangrijker - kunnen we, als alle getallen aan de beurt geweest zijn, nu niets meer doen. Het programma wordt immers afgebroken omdat de MSX een opdracht (READ) moet uitvoeren die hij niet kán uitvoeren.
We kunnen voor dit probleem twee oplossingen bedenken. Beide oplossingen hebben we al eerder gebruikt bij het typen van rijtjes getallen.

a. We kunnen de MSX van tevoren vertellen hoeveel getallen er gelezen moeten worden. We kunnen in het programma dan tellen hoeveel getallen we al gehad hebben en zo de MSX op tijd laten stoppen met het uitvoeren van de READ-opdracht.
b. We kunnen achter het rijtje getallen een speciaal getal zetten dat zelf niet meetelt. Met een vergelijking kunnen we dan het einde van de rij getallen in de dataregels herkennen.

Voeg nu de volgende regels aan het programma toe en kijk hoe het dan werkt:

```
25 IF A=9999 THEN PRINT "KLAAR": STOP
60 DATA 9999,30,47,28
```

Het eerste getal op het scherm is de 3 uit dataregel 50. Het laatste getal op het scherm is 63, ook nog van dataregel 50. Het eerstvolgende getal dat de MSX leest is de 9999. Op regel 25 staat dat het programma moet stoppen zodra de MSX dit getal tegenkomt. Dat doet hij en de getallen op dataregel 60 die na 9999 komen worden niet meer gelezen.
In plaats van het programma te laten stoppen bij het getal 9999 kunnen we de MSX ook door laten werken. Dat zien we in:

```
10 CLS
20 A=0
30 S=0
40 READ G
50 IF G=9999 THEN 100
60 PRINT G
70 A=A+1
80 S=S+G
90 GOTO 40
```

```
100 PRINT "AANTAL GETALLEN" A
110 PRINT "SOM" S
120 PRINT "GEMIDDELDE" S/A
130 STOP
500 DATA 4,-5,23,7,-8,-2,19,24,9999
```

Achter de rij getallen in de dataregel 500 staat een speciaal getal dat er zelf niet bij hoort en alleen het einde van de rij aangeeft. Als de MSX alle getallen gelezen en verwerkt heeft ontdekt hij dit speciale getal en gaat verder met regel 100. Daar zet hij de uitkomsten van het werk op het scherm.

Opmerking: We hebben nu getallen op dataregel 500 gezet. We hebben dan wat ruimte tussen het einde van het programma op regel 130 en de gegevens op regel 500. Als we het programma moeten uitbreiden hebben we daarvoor nog voldoende regelnummers over.

Voeg nu de volgende regels aan het programma toe:

```
130 GOTO 20
510 DATA 2,9,-3,7,-22,-3,9,9999
520 DATA 4,8,12,16,9999
530 DATA 22,21,20,19,18,17,9999
```

We hebben nu een programma dat rijtjes getallen leest en van elke rij het aantal getallen, de som en het gemiddelde op het scherm zet. Elk rijtje getallen eindigt met 9999.

Opmerkingen:

1. Door deze verandering van het programma zitten we nu weer met het oude probleem van OUT OF DATA. Hoe lossen we dat op?
2. Elke rij getallen staat op een nieuwe dataregel. Dat hoeft niet. We mogen op een dataregel zo veel getallen en teksten zetten als er maar op kunnen. Dat betekent dat we op elke regel 255 tekens kwijt kunnen. Bij die tekens rekenen we ook het woord DATA zelf en de komma's die tussen de getallen staan.
3. Bij het typen van dataregels kunnen we gewoon doorgaan tot 255 tekens. De MSX waarschuwt ons niet als we teveel tekens typen. We kunnen dat alleen zien aan het verspringen van de cursor. Het "teveel" wordt door de MSX gewoon niet meegenomen. We hoeven ons van de breedte van de regels op het scherm niets aan te trekken. Typ gewoon door, ook als een getal bijvoorbeeld half op de eerste regel en half op de tweede regel op het scherm komt te staan.

81. AUTO

Onze programma's worden nu langzamerhand wat langer. Het wordt daarom tijd dat we het ons wat gemakkelijker maken en de MSX te hulp roepen bij het typen van de programmaregels.
Met de opdracht AUTO zeggen we tegen de MSX dat hij voor ons de regelnummers maar vast op het scherm moet zetten, zodat wij de opdrachten erachter kunnen typen. Elke keer als wij een opdracht klaar hebben komt hij met het volgende regelnummer. Probeer dat eerst zo uit:

Wij: **NEW**
     AUTO
MSX: **10**
Wij: **CLS**
MSX: **20**
Wij: **READ A$**
MSX: **30**
Wij: **CTRL-STOP**



stop de auto

De opdracht AUTO zorgt ervoor dat de MSX met regelnummer 10 komt. Daar begint hij AUTOmatisch mee. Achter regel 10 zetten wij een PRINT-opdracht die we afsluiten met RETURN.
Zodra we RETURN gebruiken komt de MSX met regel 20. Wij vullen regel 20 in. Zodra we afsluiten met RETURN vraagt de MSX om regel 30.
Dit werkt "eindeloos" zo door. Elke keer als wij een opdracht afsluiten geeft de MSX een nieuw regelnummer. Alleen het gebruik van de CTRL-STOP laat hem daarmee ophouden.
We kunnen nu bekijken hoe het programma eruit ziet:

Wij: **LIST**
MSX: **10 CLS**
     20 READ A$

We gaan het programma uitbreiden. De eerstvolgende opdracht moet terechtkomen op regel 30. Dat kunnen we in de AUTO-opdracht aangeven op de volgende manier:

Wij: **AUTO 30**
MSX: **30**
Wij: **PRINT A$**
MSX: **40**
Wij: **GOTO 20**
MSX: **50**
Wij: **DATA JAN,PIET,HENK**
MSX: **60**
Wij: **CTRL-STOP**

We hebben de opdracht AUTO op twee manieren gebruikt:

Alleen AUTO	begin bij 10, met 10 omhoog.
Met regelnummer: AUTO 30	begin bij 30, met 10 omhoog.

In de opdracht AUTO kunnen we ook opgeven hoe groot de stap moet zijn. AUTO 500,2 betekent: begin bij 500, met 2 omhoog.

Met AUTO 20 vragen we de MSX om regels te nummeren vanaf 20. Dat doet hij, maar achter de 20 komt een sterretje als waarschuwing. De MSX geeft ermee aan dat in het programma al een regel 20 bestaat. Als we nu een nieuwe regel typen verdwijnt daarmee de oude regel 20. Als we dat niet willen (of niet zeker zijn van wat er op die regel staat) moeten we CTRL-STOP gebruiken.

Opmerkingen:
Als we bij een bestaande regel alleen RETURN aanslaan gebeurt er niets. De oude regel blijft gewoon staan.

Denk aan het stoppen met CTRL-STOP. Als we dat vergeten kan er bijvoorbeeld het volgende gebeuren:

Wij: **AUTO 60**
MSX: **60**
Wij: **GOTO 10**
MSX: **70**
Wij: **RUN** (CTRL-STOP vergeten!)
MSX: **80**
Wij: **CTRL-STOP**

Als regel 70 aan de beurt is vergeten we te stoppen en geven de opdracht RUN. Deze opdracht RUN komt nu op programmaregel 70 terecht. De MSX gaat door met regelnummers geven: 80 komt op het scherm. In het programma zien de regels 60 en 70 er nu zo uit:

60 GOTO 10
70 RUN

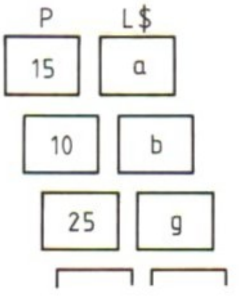
Zo'n vergissing als deze kunnen we gemakkelijk maken. Hij komt bijvoorbeeld voor bij de opdrachten RUN en LIST. Een vergissing is niet erg, als we er maar aan denken de verkeerde regels weer uit het programma te halen. Als we dat niet doen krijgen we bij het uitvoeren van het programma problemen.

Opmerkingen:
1. Een RUN-opdracht in het programma zorgt ervoor, dat de MSX weer opnieuw met het programma begint!
2. De opdracht AUTO zit ook onder functietoets F2. Deze toets zet het woord auto op het scherm, waarna wij een regelnummer kunnen invullen of met RETURN kunnen afsluiten.

82. DATA in groepen en soorten

In het volgende programma lezen we op regel 20 twee gegevens uit de dataregel. De MSX begint ook nu op de eerste dataregel vooraan. Hij leest daar een getal en een letter. Het getal komt in P, de letter in L$. In de tweede ronde komt het derde gegeven (een getal) uit de dataregel in P, het vierde (een letter) in L$. Zo worden de dataregels afgewerkt. Regel 60 zorgt ervoor dat de MSX daarna weer vooraan op de eerste dataregel begint. Zet het programma in het geheugen en laat het werken.

```
10 CLS
20 READ P,L$
30 IF P=0 THEN 60
40 PRINT TAB(P) L$
50 GOTO 20
60 RESTORE
70 GOTO 20
500 DATA 15,a,10,b,25,g,20,h,11,q
510 DATA 5,m,29,c,36,e,22,f,0
```



In dit programma zit een fout. Het programma wordt maar één keer uitgevoerd en eindigt met het bericht Out of DATA. Dat betekent dus dat er te weinig gegevens op de dataregels staan. Toch hadden we op regel 30 erop gerekend dat het programma verder zou gaan met regel 60 na het laatste getal, dus bij de 0. Hoe kan dit?

Op regel 20 lezen we steeds twee gegevens ("data"). In de dataregels horen ook steeds twee gegevens bij elkaar. Elk paar bestaat uit een positie (een getal) en een letter:

Positie,	letter,	positie,	letter,	positie,	enz.	
15 ,	131 ,	10 ,	129 ,	25 ,	enz.	
P ,	L$,	P ,	L$,	P ,	L$	

De opdracht op regel 20 leest steeds zo'n paar, een positie en een letter. Maar aan het einde van de rij lukt dat niet meer: er ontbreekt een letter.

Positie,	letter,	positie,	letter
22 ,	135 ,	0	
P ,	L$,	P ,	L$

Op regel 20 moet de MSX het laatste paar lezen, maar heeft dan één gegeven te weinig. Daarom krijgen we het bericht Out of DATA. Het programma wordt daarbij afgebroken.
Als we paren lezen moeten we er dus voor zorgen dat ook alle paren in de dataregels compleet zijn. In dit geval kunnen we dat gemakkelijk zo oplossen:

520 DATA X

Nu is ook het laatste paar compleet en kan het programma goed werken. Dat de ene "helft van het paar" op regel 510 staat en de andere helft op regel 520 vormt geen probleem!

In de READ-opdracht kunnen meer variabelen staan. Tussen de variabelen staat steeds een komma. In dit programma hebben we daarvan een voorbeeld gezien. We kennen deze mogelijkheid trouwens ook al van de INPUT-opdracht.

De drie leesopdrachten:

20 READ P
30 READ K
40 READ T$

kunnen we combineren tot één opdracht:

20 READ P,K,T$

Maar als ons dat beter uitkomt kunnen we ze ook gescheiden houden, bijvoorbeeld zo:

20 READ P
30 IF P=0 THEN 60
40 READ K,T$

We lezen nu dus op regel 20 één getal en als dat getal niet nul is op regel 40 nog een getal en een tekst. De lees-volgorde blijft altijd dezelfde. De MSX neemt steeds het volgende getal of de volgende tekst die in de dataregel aan de beurt is.

Het volgende programma leest groepjes van vijf gegevens en verwerkt die. Elke groep bestaat uit de volgende gegevens:

a. een naam
b. een geboortejaar
c. een lengte in centimeters
d. een gewicht in kilogrammen
e. het geslacht, aangegeven met één letter (V = vrouw, M = man).

Deze gegevens lezen we in de volgende variabelen:

DATA	naam,	geboortejaar,	lengte,	gewicht,	geslacht
Voorbeeld	WIM ,	1946 ,	173 ,	84 ,	M
Variabele	N$,	J ,	L ,	G ,	G$

In de eerste versie van dit programma hebben we de volgende plannen:

a. We tellen het aantal personen, het aantal mannen en het aantal vrouwen.
b. We berekenen de gemiddelde leeftijd, de gemiddelde lengte en het gemiddelde gewicht van de mannen en van de vrouwen. We gebruiken daarvoor de volgende variabelen:

	Mannen	Vrouwen
Aantal	M	V
Som leeftijden	JM	JV
Som gewichten	GM	GV
Som lengten	LM	LV

Opmerking: in het programma zijn steeds hoofdletters gebruikt. Dat hoeft niet. Houd er echter rekening mee, dat voor de MSX een A anders is dan een a. Als op regel 130 dus IF N$="EIND" staat, moet op regel 530 ook EIND staan en niet b.v. eind.

```
AUTO 110
110 M=0:V=0:JM=0:JV=0:LM=0:LV=0:GM=0:GV=0
120 READ N$
130 IF N$="EIND" THEN 270
140 READ J,L,G,G$
150 PRINT N$ " " J;L;G" "G$
160 IF G$="V" THEN 220
170 M=M+1
180 LM=LM+L
190 GM=GM+G
200 JM=JM+1985-J
210 GOTO 120
220 V=V+1
230 LV=LV+L
240 GV=GV+G
250 JV=JV+1985-J
260 GOTO 120
270 PRINT "AANTAL PERSONEN:"M+V
280 PRINT
290 PRINT "MANNEN:"M
300 PRINT "GEM.LEEFTIJD:" JM/M "JAAR"
310 PRINT "GEM.LENGTE:" LM/M "CM"
320 PRINT "GEM.GEWICHT:" GM/M "KG"
330 PRINT
340 PRINT "VROUWEN:"V
350 PRINT "GEM.LEEFTIJD:" JV/V "JAAR"
360 PRINT "GEM.LENGTE:" LV/V "CM"
370 PRINT "GEM.GEWICHT:" GV/V "KG"
380 PRINT
390 STOP   (het woord, niet de toets!)
```

Aan dit programma moeten we nu nog de dataregels toevoegen:

```
AUTO 500
500 DATA JAN,1946,172,70,M
510 DATA GREET,1950,155,50,V
520 DATA WIM,1944,180,90,M
530 DATA TRUUS,1952,160,56,V
530 DATA EIND
```

We kunnen nu eventuele fouten uit het programma halen. De MSX spoort ze voor ons op als we een RUN-opdracht geven. Als alle fouten uit het programma zijn, moeten de uitkomsten er zo uitzien:

Aantal personen: 4

	Mannen	Vrouwen
Aantal	2	2
Gem. leeftijd	35	30
Gem. lengte	176	157.5
Gem. gewicht	80	53

Opmerking: Op het scherm staan deze resultaten in een andere vorm, maar de getallen moeten wel gelijk zijn. We zetten het programma, als het helemaal klaar is, op cassette onder de naam Tellen. Denk eraan: eerst goed instellen, de opnameknop indrukken en dan de opdracht CSAVE"Tellen" geven.

83. LIST en REM

Het programma dat we nu gemaakt hebben bevat zoveel regels dat het niet meer in één keer op het scherm kan. Als we het programma willen bekijken, zullen we dat in stukjes moeten doen. Dat kan door in de LIST-opdracht regelnummers te vermelden.

Bijvoorbeeld:

LIST 100-270	(regel 100 t/m 270)
LIST -270	(van het begin tot en met regel 270)
LIST 270-	(van regel 270 tot het eind)
LIST 500	(alleen regel 500)

Als we op zoek gaan naar een bepaalde opdracht maar het regelnummer niet kennen, kunnen we het hele programma laten passeren met een gewone LIST-opdracht. Als we de gewenste regel gevonden hebben kunnen we de LIST-opdracht stoppen met CTRL-STOP.

Opmerking: Als we een MSX met een printer ("drukker") hebben, kunnen we de lijst van het programma op papier krijgen. We doen dit met de opdracht **LLIST.** Deze opdracht heeft precies dezelfde mogelijkheden als de gewone LIST opdracht. Lees eerst de paragrafen in het boekje bij de machine over het gebruik van de printer.

Een programma wordt veel beter leesbaar als we er wat toelichting in opnemen. De MSX kent een speciale opdracht waarmee dat kan. We kunnen de toelichting op aparte regels schrijven. Die regel begint dan met de opdracht REM. Alles wat op die regel staat beschouwt de MSX als toelichting op het programma, bijvoorbeeld:

```
10 REM MANNEN-VROUWEN-PROGRAMMA
15 REM DIT PROGRAMMA VERWERKT GEGEVENS
16 REM OP DATA-REGELS
```

De opdracht met de daarbij behorende toelichting mag ook op een regel staan waar al gewone opdrachten staan. We moeten er dan wel voor zorgen dat de REM achteraan op de regel staat omdat de MSX alles wat na de opdracht REM op deze regel komt als commentaar beschouwt.

170 M=M+1:REM MANNEN TELLEN

Tussen de gewone opdracht en de opdracht REM moet een dubbele punt staan. Een dubbele punt na de opdracht REM telt niet. Alles wat achter REM staat is toelichting. De PRINT-opdracht in het volgende voorbeeld wordt dus niet door de MSX uitgevoerd maar als toelichting beschouwd.

50 M=M+2:REM TWEE BIJTELLEN:PRINT M

Met duidelijke toelichting op de juiste plaats kunnen we het programma voor onszelf en voor een ander veel beter leesbaar maken.

De opdracht REM komt ons goed van pas bij het opsporen van een probleem dat nog in ons MANNEN-VROUWEN-telprogramma zit. Met de opdracht REM kunnen we namelijk een opdracht in het programma uitschakelen zonder dat we de hele programmaregel weg hoeven te halen (en misschien later weer invoegen). We kunnen het programma dan even zonder die opdracht proberen en later gemakkelijk de opdracht weer terugzetten. We doen dat door vóór de opdracht REM te zetten. We gaan dat nu doen met twee dataregels, regel 500 en regel 520. Gebruik de toets INS.

500 REMDATA JAN,1946,172,70,M
520 REMDATA WIM,1944,180,90,M

Vergeet bij dit editten de toets RETURN op elke gewijzigde regel niet!

We hebben regel 500 en 520 daarmee veranderd in toelichting. Deze twee regels tellen nu niet meer als data mee. Aan het einde van deze paragraaf zullen we de twee REM-opdrachten weer weghalen, zodat regel 500 en 520 weer gewone dataregels worden.

De bedoeling is te laten zien dat het programma niet goed werkt als er geen gegevens over mannen (of over vrouwen) in de dataregels staan. Geef een opdracht RUN en kijk wat er gebeurt.

MSX: **Division by zero in 300**

Dat is geen onbekend bericht: de MSX mag niet delen door nul. In dit programma wordt hem dat toch gevraagd omdat er toevallig geen gegevens over mannen in de dataregels zitten. Het aantal mannen blijft nul, zodat de MSX op regel 300 moet delen door 0 en dat weigert hij. Hij breekt het programma af en maakt melding van de fout. Dit is een altijd terugkerend probleem als er in een programma gedeeld moet worden door een getal waarvan de waarde niet vaststaat. Als we het risico lopen dat dat getal toevallig nul is, blijft of wordt, moeten we daarmee bij het maken van het programma rekening houden.

In een eerder voorbeeld hebben we dat gedaan door een vergelijking met nul in het programma op te nemen. We lieten de MSX het delen dan overslaan.
Dat is niet zo'n prettige oplossing als we er bij **elke** deling opnieuw aan moeten denken en steeds weer opnieuw zo'n vergelijking in het programma moeten opnemen. Daarom kiezen we nu voor een andere oplossing.

84. Foutroutine

Die oplossing heet foutroutine en we maken hem met twee opdrachten:

ON ERROR GOTO ...
RESUME NEXT

Met de eerste van deze twee opdrachten maken we de MSX duidelijk waar hij in het programma verder moet gaan als hij een programmafout tegenkomt. Op de punten moet daarom een regelnummer staan.
Als de MSX deze opdracht uitgevoerd heeft onthoudt hij bij welke regel hij verder moet gaan als hij een fout tegenkomt. Dit blijft gelden totdat de MSX een andere ON ERROR-opdracht uitvoert.
We kunnen de opdracht dus bijvoorbeeld vooraan in het programma zetten:

90 ON ERROR GOTO 1000

In deze opdracht op regel 90 staat dat de MSX bij constatering van een fout naar regel 1000 moet gaan. Daar moeten we vermelden wat de MSX moet doen bij een fout. Wij kunnen hem dan bijvoorbeeld op onze eigen manier melding laten maken van de fout, een berekening laten uitvoeren of iets dergelijks.

Bovendien moeten we de MSX vertellen waar hij dan verder moet gaan.
Daarvoor dient de tweede opdracht:

RESUME NEXT

Deze opdracht zegt dat de MSX door moet gaan met de eerstvolgende opdracht na de opdracht waarbij de fout voorkwam. We kunnen de RESUME NEXT-opdracht dus ongeveer vertalen in "ga nu gewoon verder met het programma".
Wij voegen nu het volgende aan ons programma toe:

1000 PRINT "*"**
1010 RESUME NEXT

Met de aanvullingen op regel 90, 1000 en 1010 moet het programma nu goed werken, ook als het aantal mannen of vrouwen nul is. Geef een opdracht RUN en kijk of dat waar is. De REM-opdrachten in regel 500 en 520 kunnen er daarna uitgehaald worden.

Bij elke fout gaat de MSX "even langs" bij een speciaal stukje programma. Dit stukje programma (hier de regels 1000-1010) noemen we een foutroutine. De ON ERROR opdracht op regel 90 zegt waar de foutroutine staat. De foutroutine eindigt altijd met een opdracht RESUME.

Opmerkingen:

a. De opdracht RESUME heeft meer mogelijkheden. Zie daarvoor het BASIC-boekje.
b. In het programma kunnen meer ON ERROR-opdrachten en meer foutroutines staan die betrekking hebben op verschillende soorten fouten. De MSX onthoudt altijd alleen de ON ERROR-opdracht die hij **het laatst uitgevoerd heeft.** We moeten er in het programma dus voor zorgen dat de MSX "altijd weet waar hij zijn moet" bij een fout.
c. ERR is de naam van de variabele, waarin de MSX het nummer van een fout (de foutcode) vastlegt. De MSX onthoudt ook het regelnummer waarop de fout optrad, namelijk in de variabele ERL (van ERror Line). Deze variabelen kunnen we op het scherm zetten, bijvoorbeeld met:

   1010 PRINT "Fout" ERR "op regel" ERL:RESUME NEXT

   De variabelen ERR en ERL mogen we niet zelf gebruiken. Het zijn zogenaamde "gereserveerde woorden".
   De betekenis van de foutcodes (bijvoorbeeld code 2 = syntax error) staat in het BASIC-boekje bij de computer.
d. Als de MSX eenmaal een foutroutine (ON ERROR GOTO) is tegengekomen, gaat hij bij alle fouten naar deze routine. Dus ook bij syntax-errors. Die krijgen we dan niet meer te zien. Wacht daarom met het programmeren van de foutroutine tot het programma geen syntax-errors meer bevat, of programmeer in de foutroutine het afdrukken van een "eigen foutmelding".

85. Subroutines

In de vorige paragraaf hebben we kennisgemaakt met de foutroutine. Een routine is een afgerond stuk programma. In deze paragraaf gaan we gebruikmaken van een ander soort routine, de subroutine met GOSUB en RETURN.

De opdrachten GOSUB en RETURN horen net zo bij elkaar als de ON ERROR en RESUME. De GOSUB zegt de MSX dat hij "even langs" moet gaan bij een stuk programma (routine), de RETURN zegt dat hij terug moet naar de plaats vanwaar hij gekomen is.

We gaan nu een programma maken dat ons laat zien hoe subroutines werken. In het programma zitten, naast de GOSUB- en RETURN-opdrachten, alleen maar PRINT-opdrachten die op het scherm zetten op welke regel de MSX bezig is. Typ het programma, controleer het zorgvuldig en voer het dan uit. Daarna zullen we in dit programma een paar veranderingen aanbrengen waardoor het niet meer goed werkt. We kunnen daardoor zien waaraan we moeten denken bij het gebruik van subroutines.

Maak het geheugen leeg en typ het volgende programma. Let op de puntkomma's en de dubbele punten!

```
10 R$="10":GOSUB 900
20 R$="20":GOSUB 900
30 R$="30 GOSUB 500-RETURN IS 40":GOSUB 900:GOSUB 500
40 R$="40":GOSUB 900
50 R$="50":GOSUB 900
60 R$="60 GOSUB 520-RETURN IS 70":GOSUB 900:GOSUB 520
70 R$="70 GOSUB 800-RETURN IS 80":GOSUB 900:GOSUB 800
80 R$="80":GOSUB 900
90 R$="90 GOSUB 540-RETURN IS 100":GOSUB 900:GOSUB 540
100 R$="100 STOP":GOSUB 900:STOP

500 R$="500":GOSUB 900
510 R$="510":GOSUB 900
520 R$="520 GOSUB 700-RETURN IS 530":GOSUB 900:GOSUB 700
530 R$="530 GOSUB 800+RETURN IS 540":GOSUB 900:GOSUB 800
540 R$="540 RETURN":GOSUB 900:RETURN

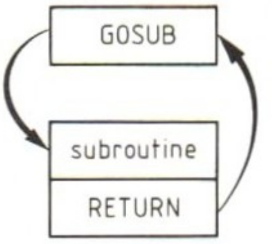
700 R$="700":GOSUB 900
710 R$="710 GOSUB 800-RETURN IS 720":GOSUB 900:GOSUB 800
720 R$="720 RETURN":GOSUB 900:RETURN

800 R$="800":GOSUB 900
810 R$="810 RETURN":GOSUB 900:RETURN

900 PRINT "NU BIJ " R$ TAB(35);:INPUT K$:RETURN
```

Een korte toelichting, voordat we het programma aan een aantal experimenten onderwerpen:

a. De opdracht GOSUB is een sprong (net als GOTO). De MSX zorgt er echter voor, dat hij onthoudt waar hij vandaan gekomen is. Dat noemen we het terugkeeradres. Hij kan een hele reeks van deze terugkeeradressen onthouden.
b. De opdracht RETURN zegt: ga naar het terugkeeradres, dat je als laatste bij een GOSUB onthouden hebt. GOSUB- en RETURN-opdrachten moeten dus altijd in evenwicht zijn (net als FOR en NEXT)!



Voer het programma uit en volg het verloop. Sla bij het vraagteken (door de INPUT op regel 900) alleen RETURN aan.

Opmerking: het programma vertelt niet precies wat het doet: we kunnen wel het verloop volgen, maar het uitstapje naar de subroutine op 900 meldt het niet steeds op het scherm.

We gaan nu de gevolgen van een reeks wijzigingen in het programma bestuderen. Breng de wijziging aan, voer het programma uit en maak daarna de wijziging weer ongedaan.

a. Verwijder van regel 60 de GOSUB 520 achteraan. Conclusie: geen probleem (wel weer terugzetten! Vergeet de dubbele punt niet.)
b. Verwijder van regel 810 de RETURN achteraan. Conclusie: zonder return gaat de MSX door naar regel 820. Wat er dan gebeurt hangt af van de rest van het programma. Zet de RETURN er weer in.
c. Haal de RETURN van regel 720 achteraan weg
d. Zet op regel 45 GOTO 800
e. Zet op regel 805 GOTO 530
f. Zet op regel 805 GOTO 520

Opmerkingen:
a. Wees niet te snel met de conclusie, dat een verandering geen probleem is. Soms moet een programma vele malen uitgevoerd worden, voordat een fout blijkt!
b. Bij het foutbericht "RETURN without GOSUB" gaat het er niet om, of de opdrachten GOSUB en RETURN in het programma staan en in evenwicht zijn, maar om de vraag of de MSX ze "in evenwicht" kan uitvoeren!

Subroutines hebben twee functies:

a. We kunnen het programma overzichtelijker maken door delen van het programma bij elkaar te nemen en in een subroutine onder te brengen.
b. We kunnen een programmadeel, dat op een aantal plaatsen in het programma uitgevoerd moet worden slechts éénmaal in het programma opnemen en er vanaf elke plaats waar dit deel nodig is met een GOSUB "even langs gaan". De subroutine op regel 900 in het vorige programma is daarvan een goed voorbeeld. Zonder subroutine hadden we de opdrachten op regel 900 steeds moeten herhalen (de sprong naar 900 komt op elke programmaregel voor).

Er is één fout, waarvoor we uitdrukkelijk moeten waarschuwen:

Spring nooit met een GOTO uit een subroutine, maar altijd en uitsluitend met een RETURN.

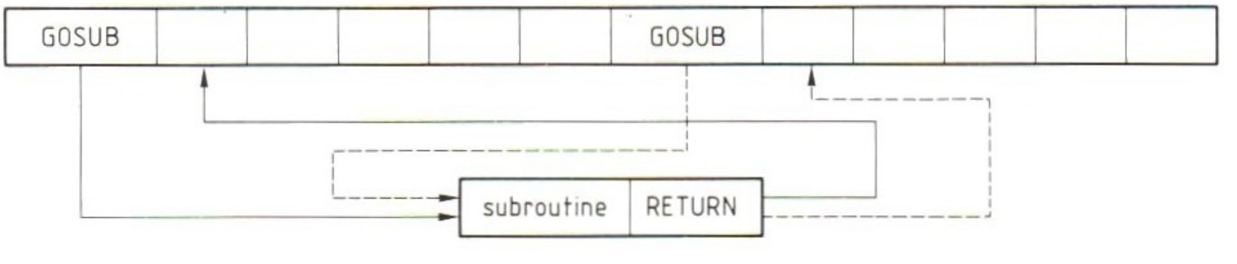
Het volgende programma laat zien, dat deze stelregel echt opgaat, maar dat de gevolgen pas blijken als de MSX het programma ruim 4000 keer achter elkaar uitgevoerd heeft:

```
10 R=0
20 PRINT R
30 GOSUB 100
40 PRINT "KLAAR"
50 STOP
100 R=R+1
110 GOTO 20
120 RETURN
```

Typ het programma, geef RUN en wacht op **Out of memory in 20,** wat zoveel wil zeggen als: het geheugen is vol!

Deze fout wordt veroorzaakt door het feit, dat de MSX elke keer als hij op regel 30 de GOSUB uitvoert het bijbehorende terugkeeradres (regel 40) moet onthouden. Die terugkeeradressen stapelen zich maar op, omdat er nooit één afgaat: de MSX komt geen RETURN tegen. Pas als hij een RETURN <u>uitvoert</u> mag hij het laatst onthouden terugkeeradres "vergeten".

Opmerking: de 400 keer geldt bij een geheugen van ruim 28000 tekens; bij een geheugen van ± 12000 tekens komt de MSX niet verder dan 1700 keer.



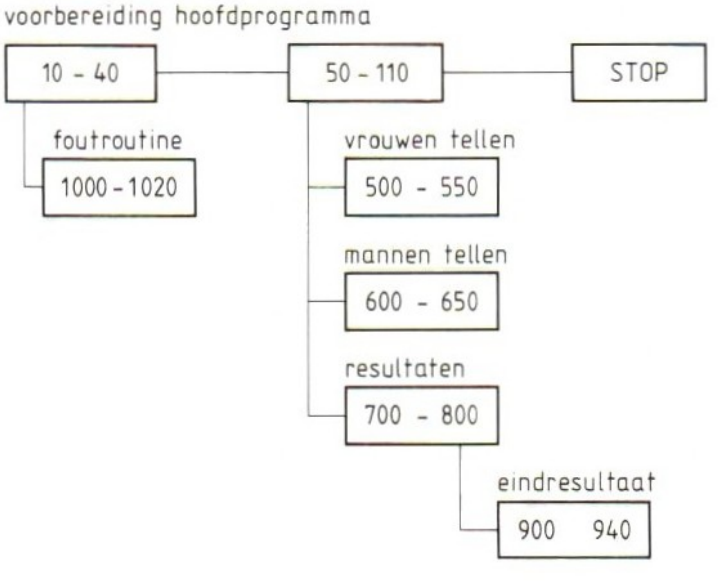
86. Mannen en vrouwen tellen

In deze paragraaf hebben we het programma dat mannen en vrouwen telt opnieuw opgezet en gebruikgemaakt van subroutines. Het programma is daardoor en door het toevoegen van toelichting duidelijker geworden. De ** in REM-opdrachten dienen alleen om ze op te laten vallen!

Neem het programma over in de MSX en probeer het uit. Leg het daarna vast op de cassette met de opdracht CSAVE.

Opmerkingen:

a. Een tekst die in een programma meer dan één keer voorkomt, kan beter in een variabele gezet worden. In dit programma is dat gedaan op regel 30.
b. Op regel 770 is een trucje uitgehaald om ervoor te zorgen dat de resultaten van de mannen en de vrouwen met dezelfde subroutine op regel 900 afgedrukt kunnen worden.
   Voordat we de subroutine binnen gaan worden de resultaten van de vrouwen overgebracht naar de variabelen van de mannen.
   Het resultaat van de mannen is toch al op het scherm gezet. De "mannenvariabelen" mogen dus overschreven worden.
c. Het programma staat al op de cassette onder de naam Tellen. Dit "oude" programma mag door het nieuwe overschreven worden.



```
10 REM**TELPROGRAMM**
20 ON ERROR GOTO 1000
30 A$="GEM."
40 M=0:V=0:JM=0:JV=0:LM=0:LV=0:GM=0:GV=0
50 REM**HOOFDPROGRAMMA**
60 READ N$
70 IF N$="EIND" THEN GOSUB 700:STOP
80 READ J,L,G,G$
90 PRINT N$ " "J;L;G" "G$
100 IF G$="V" THEN GOSUB 500 ELSE GOSUB 600
110 GOTO 50
500 REM**TELLEN VROUWEN**
510 V=V+1
520 LV=LV+L
530 GV=GV+G
540 JV=JV+1982-J
550 RETURN
600 REM**TELLEN MANNEN**
610 M=M+1
620 LM=LM+L
630 GM=GM+G
640 JM=JM+1982-J
650 RETURN
700 REM*RESULTATEN*
710 CLS
720 PRINT "AANTAL PERSONEN"M+V
730 PRINT
740 PRINT "MANNEN:"M
750 GOSUB 900
760 PRINT
770 M=V:LM=LV:GM=GV:JM=JV
780 PRINT "VROUWEN:"M
790 GOSUB 900
800 RETURN
900 REM**EINDRESULTAAT**
910 PRINT A$"LEEFTIJD:" JM/M "JAAR"
920 PRINT A$"LENGTE:" LM/M "CM"
930 PRINT A$"GEWICHT:" GM/M "KG"
940 RETURN
1000 REM*FOUTROUTINE*
1010 PRINT "DELEN DOOR 0"
1020 RESUME NEXT
5000 DATA JAN,1946,172,70,M,WIM,1944,180,90,M
5010 DATA GREET,1959,155,50,V,TRUUS,1952,160,56,V
5020 DATA DORY,1943,168,65,V,WILLEM,1938,181,84,M
5030 DATA GEERT,1941,184,81,M,WILMA,1949,158,52,V
5040 DATA EIND
```

87. Opdrachten

Opdracht 1:
Sommige opdrachten vormen een duidelijk, soms zelfs "onafscheidelijk paar".
Vul in:

Bij: GOSUB hoort ....

ON ERROR ....

FOR ....

DATA .... en ....

IF .... en ....

Opdracht 2:
Dataregels kunnen erg handig zijn bij het testen van een programma. We nemen dan gegevens die eigenlijk getypt moeten worden bij een INPUT-opdracht eerst op in dataregels om het programma uit te proberen. Dat scheelt veel typwerk. Neem het volgende programma over in de MSX:

```
10 CLS
20 INPUT "BEGINSTAND TELLER";B
30 INPUT "EINDSTAND TELLER";E
40 K=E-B
50 PRINT "VERREDEN" K "KM"
60 PRINT
70 PRINT "GETANKT"
80 INPUT L
90 IF L=0 THEN 120
100 T=T+L
110 GOTO 80
120 PRINT "AANTAL KM/LITER:" K/T
```



Dit programma vraagt op regel 20 en 30 om de begin- en eindstand van de kilometerteller. Daarna kunnen we op regel 80 van alle keren dat we getankt hebben het aantal liters typen. De MSX telt dat bij elkaar op en rekent uit hoeveel het benzineverbruik is. Als we dit programma willen uitproberen kunnen we een opdracht RUN geven en de door de MSX gevraagde gegevens typen.

Om te weten of het programma goed werkt zullen we dat een paar keer moeten doen. We kunnen dit gemakkelijker maken door een reeks "testgegevens" op dataregels te zetten. We moeten dan het programma wel aanpassen.
De INPUT-opdrachten moeten vervangen worden door READ-opdrachten. Breng deze veranderingen aan en zet de testgegevens erbij op dataregels. Maak het programma en de testgegevens zo dat we zonder typwerk kunnen zien hoe het programma reageert op de ingevoerde gegevens, ook als er fouten in die gegevens zitten.

Aanwijzing: Een fout is bijvoorbeeld een eindstand die lager is dan een beginstand. Is een negatief ingevoerd aantal liters op regel 80 ook fout?

Opdracht 3:
In de getallen en teksten op dataregels kunnen we ook op zoek gaan naar een bepaald getal of een bepaalde tekst. Dat kan echter alleen in volgorde. In het computerjargon heet dat 'sequentieel'. Het betekent dat we elke keer vooraan op de dataregels moeten beginnen en net zo lang moeten lezen tot we het gezochte gegeven gevonden hebben.
Het volgende programma is daarvan een voorbeeld:

```
10 CLS
20 INPUT "MAAND";M
30 FOR A=1 TO M
40 READ A$
50 NEXT
60 PRINT "MAAND"M"IS "A$
70 PRINT
80 GOTO 20
```

In dit programma kunnen we bijvoorbeeld de zesde maand van het jaar opzoeken door zes maal te lezen op regel 40. Na de zesde keer stoppen we, zodat in A$ de naam JUNI staat. Voeg aan dit programma de dataregel of -regels toe.
Als we het programma meer dan één keer willen uitvoeren, moet er ook een RESTORE opdracht toegevoegd worden. Zet deze opdracht op de juiste plaats.
Zoeken op dataregels op deze manier (en een andere manier is er niet) kan wel wat tijd gaan kosten als er veel gegevens op dataregels zijn waarin gezocht moet worden. Probeer dat uit door aan de dataregel in dit programma nog een groot aantal gegevens toe te voegen, zodat de MSX bijvoorbeeld op zoek moet naar het 50ste gegeven.

88. ASCII-codes

In dit hoofdstuk zullen we wat verder ingaan op de mogelijkheden die we hebben bij het programmeren met strings. We hebben de string in de vorige hoofdstukken al op verschillende manieren gebruikt, bijvoorbeeld in de PRINT-opdracht, als gegeven op een dataregel, bij een INPUT-opdracht (op twee manieren zelfs!)
We kennen ook de stringvariabele die herkenbaar is aan het dollarteken.
In dit hoofdstuk zullen we kennismaken met een paar nieuwe mogelijkheden.
Daarvoor gaan we eerst kijken hoe een string in het geheugen van de MSX opgeborgen wordt.

We geven de MSX opdracht een string in het geheugen op te bergen met bijvoorbeeld:

A$="CARMEN"

Deze opdracht zegt dat de string CARMEN opgeborgen moet worden in vakje A$ of, om het netter te zeggen, de string CARMEN wordt toegekend aan de variabele A$. Als we het geheugen van de MSX open zouden kunnen maken om te kijken hoe de string opgeborgen wordt, zouden we twee dingen ontdekken:

a. Alle letters van de naam CARMEN staan in het geheugen als getallen.
   Voor de naam CARMEN zijn dat de volgende getallen:

C=67	M=77
A=65	E=69
R=82	N=78

b. A$ is wel de naam van één vak (één variabele) in het geheugen, maar dat vak bestaat uit een aantal kleinere vakjes. In elk vakje past één letter, voorgesteld door een getal. CARMEN staat dus eigenlijk zo in het geheugen:

   C A R M E N
   67-65-82-77-69-78

   Letters staan dus als getallen in de MSX en elke letter heeft zijn eigen vakje.

We kunnen de getallen die bij de letters horen gebruiken in de characterstring:
CHR$(...) (CHR van CHaRacter, teken; $ van string, dus: string van één teken).

Deze functie gaat over één letter of, beter gezegd, over één teken, want in plaats van letters kunnen we ook werken met bijvoorbeeld een uitroepteken, een vraagteken, de komma, de punt e.d. Het getal dat bij het teken hoort komt tussen de haakjes te staan. We kunnen de CHR$-functie bijvoorbeeld gebruiken in een PRINT-opdracht:

Wij: **PRINT CHR$(74)CHR$(65)CHR$(78)**
MSX: **JAN**

Elke CHR$ opdracht wijst één letter aan. Hier zijn dat de letters J A N omdat tussen haakjes de getallen staan die bij deze letters horen.
We kunnen ook letters aan elkaar knopen tot een langere string door ze "op te tellen"(maak deze regel met de edit-mogelijkheden uit de vorige!):

Wij: **A$=CHR$(74)+CHR$(65)+CHR$(78)**
     PRINT A$
MSX: **JAN**

Maak nu het geheugen van de MSX leeg en typ het volgende programma:

```
10 CLS
20 FOR I=65 TO 122
30 PRINT I " komt overeen met " CHR$(I)" ";
40 NEXT
```

Aan dit programma kunnen we een paar dingen zien:

a. Niet alle getallen stellen letters voor; er zijn ook getallen die iets anders betekenen. De 12 bijvoorbeeld hoort niet bij een letter maar betekent "maak het scherm schoon".
b. Als in een opdracht een getal mag staan mogen we daarvoor in de plaats ook een variabele of een berekening zetten. Dat geldt ook voor de CHR$. Op regel 30 blijkt dat. Op deze regel geven we de MSX opdracht het teken op het scherm te zetten waarvan het bijbehorende getal in de variabele I staat. Het getal in I zelf komt ook op het scherm te staan.

Dit programma laat zien welke letters de MSX kent en welke getallen daarbij horen. Wat het programma laat zien is een deel van de

ASCII-code (American Standard Code for Information Interchange).

Deze ASCII-code wordt in veel computers gebruikt. Voor al deze computers geldt bijvoorbeeld dat de letter A opgeborgen wordt als het getal 65.

De naam van de code, ASCII, komt terug in de volgende opdracht:

Wij: **PRINT ASC("J")**   (ASC van ASCII)
MSX: **74**

Met de opdracht ASC vragen we de MSX de ASCII-code te laten zien van de letter tussen haakjes. Deze letter is een string, dus moet hij tussen aanhalingstekens staan. De MSX vertelt ons dat bij de letter J de code 74 hoort. De ASCII-code van een letter is gewoon een getal. Dat getal kunnen we weer gebruiken in alle opdrachten waarin we met getallen werken. Dat blijkt bijvoorbeeld in het volgende programma:

```
10 CLS
20 INPUT "WELKE LETTER";A$
30 A=ASC(A$)-96
40 PRINT A$ " IS LETTER NUMMER" A
50 GOTO 20
```

Op regel 30 van dit programma laten we de MSX eerst uitzoeken wat de ASCII-code van de ingevoerde letter is. Daarna halen we van die code 96 af. LET OP! Dit programma werkt alleen goed voor kleine letters!

Als we de opdrachten **ASC(...)** en **CHR$(...)** vergelijken, zien we dat ze precies het tegenovergestelde doen. De eerste opdracht maakt van een letter een code, de tweede van een code weer een letter. We kunnen het programma dat we zojuist geprobeerd hebben dus ook omkeren. In het eerste voorbeeld lieten we de MSX uitzoeken wat het volgnummer van een letter in het alfabet is; in het volgende voorbeeld geven we een nummer en laten we de MSX uitzoeken welke letter daarbij hoort.

```
10 CLS
20 INPUT "HOEVEELSTE LETTER";N
30 IF N<1 OR N>26 THEN PRINT "FOUT":GOTO 20
40 A$=CHR$(N+64)
50 PRINT "LETTER" N"IS "A$
60 GOTO 20
```

In dit voorbeeld zien we dat we tussen de haakjes ook weer een berekening mogen zetten.

De hoofdletters hebben andere codes dan de kleine letters. Die codes liggen precies 32 uit elkaar. Vergelijk bijvoorbeeld:

A=65	a= 97
B=66	b= 98
Z=90	z=122

Daarvan maken we gebruik in het volgende programma-voorbeeld:

```
10 CLS
20 INPUT "HOOFDLETTER";A$
30 A=ASC(A$)
40 A=A+32
50 B$=CHR$(A)
60 PRINT "BIJ " A$ " HOORT " B$
70 GOTO 20
```

De drie opdrachten op regel 30, 40 en 50 kunnen we desgewenst combineren tot één opdracht. Die ziet er dan op regel 30 zo uit:

```
30 B$=CHR$(ASC(A$)+32)
```

Verklaring: ASC(A$) is de ASCII-code van A$
ASC(A$)+32 is de ASCII-code van A$, verhoogd met 32
CHR$(ASC(A$)+32) is het teken, dat hoort bij de met 32 verhoogde ASCII-code van A$.

Opmerking : Vergeet niet regel 40 en 50 weg te halen!

Vervang nu regel 30 door de volgende opdracht:

30 B$=CHR$(ASC(A$)+128)

en kijk wat het resultaat is.

Ga na welke van de volgende opdrachten fout zijn en welk bericht de MSX geeft.

a. PRINT ASC(J)
b. PRINT CHR$(J)
c. PRINT ASC("A")+32
d. PRINT ASC("A")+A$
e. PRINT CHR$(ASC("A")+32)
f. PRINT CHR$(ASC("A")+32))
g. PRINT CHR$(ASC("A"))
h. PRINT ASC("JAN")
i. PRINT CHR$(ASC("JAN"))
j. PRINT CHR$(-65)
k. PRINT CHR$(255)
l. PRINT CHR$(256)

Conclusie: Tussen de haakjes van de ASC-opdracht hoort een string te staan. Als dit een string is van meer dan één teken, geeft de MSX alleen de ASCII-code van de eerste letter. Tussen de haakjes van een CHR$ hoort een getal te staan. Dat getal mag er staan in de vorm van een constante, een variabele of een berekening met een waarde van 0 tot en met 255. De variabele mag wel weer een ASC-functie zijn of een ASC-functie in een berekening.
We moeten wel goed opletten dat de haakjes op de juiste plaats staan en dat we er geen vergeten.

89. Stuurcodes

Niet alle getallen in de CHR$-functie stellen een letterteken voor. De reeks letters en cijfers begint pas bij ASCII-code 32. Probeer:

```
10 CLS
20 FOR I=32 TO 255
30 PRINT CHR$(I);
40 NEXT
```

We zien dan alle tekens op een rijtje: eerst wat speciale tekens, dan cijfers, hoofdletters, weer wat vreemde tekens, kleine letters, letters met accenten, een deel van het Griekse alfabet, grafische tekens, enz.
In het BASIC-boekje bij de MSX staat een tabel van alle tekens en de daarbij behorende ASCII-codes.

De ASCII-codes 1 tot en met 31 zijn zogenaamde stuurcodes: we kunnen er de computer tijdens het printen mee sturen.

Maak het volgende programmaatje:

```
10 CLS
20 INPUT "CHR$";C
30 PRINT TAB(5) "Kijk mij eens!" CHR$(C) "123"
40 PRINT
50 GOTO 20
```

Op regel 20 mogen we een code ingeven, die op regel 30 als teken (character) tussen de teksten "Kijk mij eens!" en "123" op het scherm wordt gezet.

Probeer nu uit wat de codes 1 tot en met 32 doen:

Code	Resultaat
1-6	niet te zien
7	piep! (niet te zien, wel te horen)
8	zet de cursor één plaats terug, net als de BS-toets
9	zet de cursor op de volgende tabulatorpositie (zoals de toets TAB, <u>niet</u> zoals de functie TAB)
10	zet de cursor naar de volgende regel, ook als we helemaal onderaan het scherm zitten!
11	zet de cursor naar het begin van het scherm linksboven (zoals de toets HOME)
12	maakt het scherm schoon en zet de cursor linksboven (zoals de CLR-toets en de opdracht CLS)
13	zet de cursor naar het begin van de regel
14-20	niet te zien
21-26	niet te zien
27	er ontbreekt één teken van de tweede tekst
28	de cursor gaat één plaats naar rechts
29	de cursor gaat één plaats naar links
30	de cursor gaat één plaats (regel) naar boven
31	de cursor gaat één plaats (regel) naar onder

Van een aantal codes is duidelijk te zien wat ze doen, van andere niet. Dat is niet zo erg want er is nog een andere mogelijkheid om dezelfde functies te krijgen.

Daarvoor gebruiken we de toets CTRL (een afkorting van CONTROL, besturen), samen met en andere toets. De CTRL werkt zoals de toets SHIFT: indrukken, vasthouden en een andere toets aanslaan, loslaten.
We proberen dat als volgt: breek het programma af, maak het scherm schoon en geef een opdracht LIST.
Typ nu:

CTRL-l (CTRL-L mag ook)

De MSX maakt het scherm schoon, net als wanneer we de toets CLR (SHIFT-HOME) gebruiken, de opdracht CLS geven of de opdracht PRINT CHR$(12) uitvoeren.

Geef nog een keer een LIST-opdracht en probeer:

CTRL-k

Waar is de cursor gebleven?

Probeer nu:

CTRL-f (een paar keer achter elkaar)

De cursor springt bij elke CTRL-f naar het begin van het volgende woord.

CTRL-b (enkele keren)

De cursor springt naar het begin van het vorige woord. Kijk goed wat als "woord" geldt.

Zet nu de cursor op regel 30 achter het sluithaakje van TAB(5) en druk op:

CTRL-e

De rest van regel 50 verdwijnt plotseling (pas als we op RETURN drukken verdwijnt dit stuk écht uit het programma).

Zet de cursor ergens op regel 40 en typ:

CTRL-u

De hele regel verdwijnt. Ook hier geldt: pas als we op RETURN drukken, verdwijnt de regel écht uit het programma.

Zet de cursor op regel 20 en geef:

CTRL-n

De cursor springt naar het eind van de regel. We kunnen nu gemakkelijk iets toevoegen aan deze regel.

Tenslotte deze: zet de cursor op een schone regel, geef een RUN-opdrach, voer het programma een paar keer uit en druk dan op:

CTRL-c

De MSX breekt het programma af, net als wanneer we de CTRL-STOP gebruiken.

In het BASIC-boekje bij de machine staat een lijst met de mogelijkheden van CTRL in combinatie met andere toetsen.

Opmerkingen:

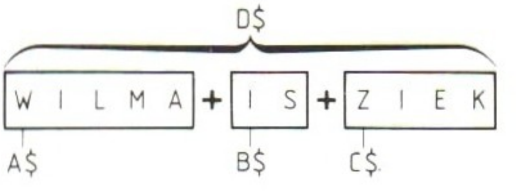
1. CTRL met een letter werkt ook bij het uitvoeren van een programma bij een INPUT-opdracht. Voer het programma uit, geef een paar codes en probeer dan bijvoorbeeld wat CTRL-b, CTRL-1, CTRL-n enz. doen.
2. De toets TAB heeft hetzelfde gevolg als de opdracht PRINT CHR$(9) of CTRL-i.
3. De opdracht BEEP heeft hetzelfde gevolg als de opdracht PRINT CHR$(7) en als de toets CTRL-g.
   Probeer:

   FOR I=1 to 10:BEEP:NEXT

90. Strings bewerken

Strings kunnen we aan elkaar koppelen door ze op te tellen. Dat doen we met een plus-teken.

```
10 CLS
20 CLEAR 500
30 A$="WILMA"
40 B$="IS"
50 C$="ZIEK"
60 D$=A$+B$+C$
70 E$=B$+A$+C$+"?"
80 PRINT D$
90 PRINT E$
100 STOP
```



Zet dit programma in de MSX en voer het uit. Merk op dat we bij het werken met strings altijd op moeten letten of er voldoende spaties staan en of we op de juiste plaats terechtkomen. Verbeter het programma.

De opdracht CLEAR gebruiken we om in het geheugen van de MSX meer ruimte te maken voor strings. Als we deze opdracht weg laten krijgen we van de MSX ruimte voor 200 tekens met de kans op "Out of string space".

In het volgende voorbeeld knopen we letters aan elkaar tot een woord of een zin. Zet het programma in de MSX en probeer het uit.

```
10 CLS
20 A$=""
30 INPUT "LETTER";B$
40 IF B$="!" THEN 70
50 A$=A$+B$
60 GOTO 30
70 PRINT A$
80 GOTO 20
```

Op regel 20 krijgt de stringvariabele A$ een beginwaarde, maar wel een merkwaardige. Achter het =-teken staan twee dubbele aanhalingstekens achter elkaar, zonder iets ertussen. Daarmee maken we van A$ een zogenaamde "lege string". Dat is een string die uit 0 tekens bestaat.

Als het programma de eerste keer uitgevoerd wordt is dat niet nodig: A$ is dan nog nooit gebruikt en is "vanzelf" een lege string, zoals een getalvariabele vanzelf de waarde 0 heeft.
Als het programma na regel 80 voor de tweede keer uitgevoerd wordt, is het wél nodig A$ leeg te maken. Kijk zelf maar wat het gevolg is als we regel 20 tijdelijk uitschakelen door er REM voor te zetten:

```
20 REM A$=""
```

Opmerking: we kunnen regel 20 niet weghalen, want dan krijgen we met de GOTO regel 80 problemen.

In het laatste voorbeeld van deze paragraaf maken we een string van hartjes. Zet het programma in de MSX en probeer het uit.

```
10 CLS
20 INPUT "AANTAL";A
30 A$=""
40 FOR I=1 TO A
50 A$=A$+ "♥"
60 NEXT I
70 PRINT A$
80 PRINT
90 GOTO 20
```

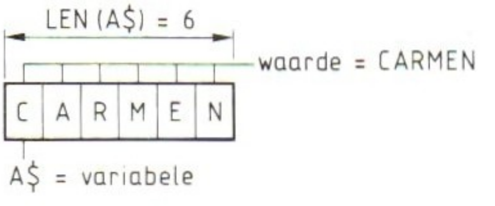
Opmerking: Kijk wat er gebeurt als A groter is dan 50. Voeg een CLEAR-opdracht toe!

Sommige MSX-tekens zijn breder dan de normale tekens. Daardoor kunnen ze niet in hun geheel op het scherm worden gezet. De computer laat aan de rechterkant een randje weg. Dat is bijvoorbeeld bij het hartje het geval.

De computer kan op het tweede schermtype wel de bredere tekens kwijt. De normale lettertekens staan op dit scherm wat verder uit elkaar.
Voeg aan het programma een opdracht SCREEN 1 toe (zoek zelf een goede plaats voor deze opdracht) en verander de kleuren met een COLOR-opdracht.

Strings zijn niet allemaal even groot. Als we bijvoorbeeld woorden typen zijn er korte en lange woorden. Soms is het belangrijk te weten uit hoeveel tekens een string bestaat. We kunnen dat de MSX laten uitzoeken met een speciale functie. Deze functie is verwerkt op regel 30 in het volgende programma:

```
10 CLS
20 INPUT A$
30 PRINT "AANTAL TEKENS IS" LEN(A$)
40 PRINT
50 GOTO 20
```



De functie waarmee we de lengte (dat is het aantal tekens) van een string kunnen laten bepalen is LEN:

LEN(naam van de string of de string zelf) (LEN van LENgth).

In het volgende voorbeeld zetten we op het scherm evenveel streepjes als het getypte woord lang is:

```
10 CLS
20 INPUT A$
30 A=LEN (A$)
40 FOR I=1 TO A
50 PRINT "-";
60 NEXT I
70 PRINT
80 GOTO 20
```

De functie LEN(A$) stelt een getal voor. We kunnen deze opdracht dus gebruiken op alle plaatsen waar een getal staat.
In dit programmavoorbeeld kunnen we regel 30 en 40 dus vervangen door:

```
40 FOR I=1 TO LEN(A$)
```

Zoek nu uit wat de MSX doet met de volgende opdrachten:

a. PRINT LEN("HOEVEEL LETTERS"),LEN(HENK)
b. PRINT LEN A$
c. PRINT LEN("JAN")+8
d. LEN(A$)=48

De conclusies zijn:

a. tussen de haakjes mag ook een string staan, maar die moet wel tussen aanhalingstekens komen;
b. de haakjes mogen niet vergeten worden;
c. met LEN kunnen we rekenen;
d. LEN kan niet "omgekeerd" worden. Hij geeft de MSX opdracht om de lengte van een bestaande string te bepalen. We kunnen er niet de lengte van een string mee veranderen.

91. Links, rechts of middenin

We gaan nu kijken naar het andere kenmerk van strings: een string bestaat uit letters die elk in een vakje staan.

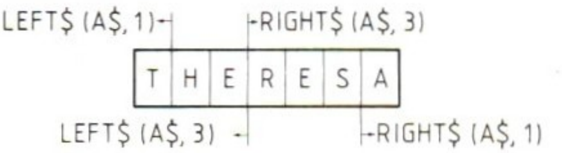
```
10 CLS
20 A$="Carmen"
30 PRINT A$
40 PRINT LEFT$(A$,1)"!"
50 PRINT LEFT$(A$,4)"!"
60 PRINT LEFT$(A$,8)"!"
70 P=5
80 PRINT LEFT$(A$,P)"!"
90 STOP                (opdracht, niet toets!)
```

Met deze nieuwe functie kunnen we een gedeelte uit de string halen vanaf links, dat wil dus zeggen vooraan uit de string. Hoeveel tekens we nemen geven we in de functie aan. De opdracht ziet er zo uit:

LEFT$(string,aantal tekens)

Op de plaats van STRING komt de naam van de string of de string zelf. Op de plaats van AANTAL TEKENS komt een getal, een variabele of een berekening, kortom weer alle mogelijkheden die we hebben om getallen aan te geven.
Het stukje string dat we met deze functie uitzoeken kunnen we op het scherm zetten met een PRINT-opdracht, maar ook in andere opdrachten gebruiken. Dat blijkt uit het volgende voorbeeld. Zoals we met LEFT$ iets aan de linkerkant, dus vooraan, uit de string kunnen halen, kunnen we met RIGHT$ iets aan de rechterkant, dus achteraan uit de string, pakken.

```
10 CLS
20 A$="THERESA"
30 PRINT A$
40 PRINT LEFT$(A$,1)
50 PRINT RIGHT$(A$,1)
60 B$=LEFT$(A$,3)
70 C$=RIGHT$(A$,3)
80 PRINT B$"R"C$
90 STOP
```



In het volgende voorbeeld hebben we de functie LEFT$ gecombineerd met de LEN. Van het woord dat we op regel 20 ingeven krijgen we eerst alleen de eerste letter, dan de eerste twee, de eerste drie, enz. tot het hele woord op het scherm staat.

```
10 CLS
20 INPUT "Woord";A$
30 FOR I=1 TO LEN(A$)
40 PRINT LEFT$(A$,I)
50 NEXT I
60 PRINT
70 GOTO 20
```

Als we de opdrachten die we in dit hoofdstuk hebben leren kennen met elkaar combineren krijgen we weer nieuwe mogelijkheden. Dat blijkt in het volgende programma:

```
10 CLS
20 READ T$,F1$,F2$,F3$
30 PRINT T$
40 INPUT A$
50 IF LEN(A$)<>5 THEN PRINT F1$:GOTO 40
60 IF ASC(LEFT$(A$,1))<97 OR ASC(RIGHTS(A$,1))>122 THEN PRINT F2$:GOTO 40
70 IF LEFT$(A$,1)<>"j" THEN PRINT F3$:GOTO 40
80 PRINT "goed"
90 PRINT
100 GOTO 40
500 DATA "Typ een woord van 5 letters, dat met een kleine letter j begint"
510 DATA "Dit zijn geen 5 letters"
520 DATA "Dit woord begint niet met een kleine letter"
530 DATA "Dit woord begint niet met j"
```

Toelichting:

a. Het programma vraagt op regel 40 om een woord. Dat moet een woord van vijf letters zijn, dat met een kleine letter j begint. Als het woord niet aan de regels voldoet, geeft het programma een bericht. Het programma geeft zo goed mogelijk aan wat er aan de hand is.
b. De teksten in het programma zijn op dataregels gezet. Deze dataregels worden op regel 20 gelezen. Deze methode is vaak handig bij het programmeren. Alle teksten in het programma staan dan bij elkaar, wat gemakkelijk is bij het wijzigen.
c. Op regel 60 gebruiken we de ASCII-code van de eerste letter van het woord om uit te maken of het woord met een kleine letter begint of niet. De kleine letters hebben codes van 97 (a) t/m 122 (z).
   De combinatie van de ASC en de LEFT$ zit zo in elkaar:

   ASC(LEFT$(A$,1))
   LEFT$(A$,1) = eerste letter van A$
   ASC(LEFT$(A$,1)) = ASCII-code van de eerste letter van A$

d. Op regel 60 staat een gecombineerde vergelijking: **als ... of ... dan ...**

Merk op dat het woord IF na OR (of) niet herhaald wordt. De MSX gaat verder achter THEN als het antwoord op één van beide vergelijkingen "ja" is.

Zet het programma in de MSX en voer het uit. Let bij het typen van het programma vooral goed op de haakjes. Als ergens een haakje vergeten wordt, levert dat bij de uitvoering direct problemen op.

We hebben al eerder gebruikgemaakt van de mogelijkheid om strings te vergelijken. Als de MSX teksten met elkaar vergelijkt, vergelijkt hij eigenlijk letter voor letter de ASCII-codes. Daarbij geldt dan bijvoorbeeld:

A (code 65) is kleiner dan B (code 66)
* (code 42) is kleiner dan A (code 65)
a (code 97) is groter dan Z (code 90)

Voor de MSX is een letter A dus kleiner dan een letter B. Dat is handig bij het sorteren van bijvoorbeeld namen. De A komt dan voor de B en zo hoort het ook. We kunnen het vergelijken van teksten bestuderen met het volgende programma:

```
5 CLEAR 500
10 CLS
20 INPUT "TWEE WOORDEN"; A$,B$
30 IF A$=B$ THEN PRINT A$" = "B$
40 IF A$>B$ THEN PRINT A$" KOMT NA "B$ ELSE PRINT A$" KOMT VOOR "B$
50 GOTO 20
```

Op regel 20 typen we steeds twee "woorden". In die woorden mogen alle tekens gebruikt worden, dus ook bijzondere tekens als streepjes, sterretjes en grafische tekens.
Alleen de komma levert problemen op, maar dat wisten we al.
Ga nu na hoe de vergelijking uitvalt als de volgende woorden getypt worden:

Typ voor:

A$	B$	
ABC	abc	
ABCD	abc	
ABC	ABCD	
A	!B!	
aBc	AbC	
100	50	(let op: vergelijking als string, niet als getal!)
A B	AB	
100	ABC	
's	s	

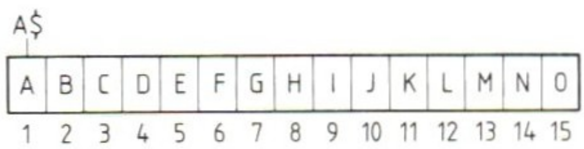
Met een derde opdracht kunnen we een gedeelte midden uit een string halen. Deze opdracht ziet er zo uit:

MID$(string,positie,aantal)

In de MID$ staat tussen de haakjes:

a. over welke string het gaat;
b. op welke positie in de string we willen beginnen;
c. hoeveel tekens we willen nemen (maar dat mogen we weglaten; we krijgen dan gewoon "de rest" van de string).

```
10 CLEAR 500
20 CLS
30 A$="ABCDEFGHIJKLMNO"
40 PRINT MID$(A$,1,5)
50 PRINT MID$(A$,5,1)
60 B$=MID$(A$,8,4)
70 PRINT B$
80 STOP
```



In dit voorbeeld staan drie MID$ opdrachten.

a. MID$(A$,1,5) is 5 tekens, te beginnen op positie 1, dus ABCDE
b. MID$(A$,5,1) is 1 teken , te beginnen op positie 5, dus E
c. MID$(A$,8,4) is 4 tekens, te beginnen op positie 8, dus HIJK

In de functie MID$ zijn "positie" en "aantal" getallen. Waar getallen staan kunnen ook getalvariabelen en berekeningen staan.

```
10 CLS
20 A$="ABCDEFGHIJKLMNO"
30 P=4:N=8
40 PRINT MID$(A$,P,N)
50 PRINT MID$(A$,N,P)
60 PRINT MID$(A$,N+2,P-3)
70 P=P+2:N=N-4
80 PRINT MID$(A$,P,N)
90 STOP
```

Vul nu eerst de volgende regels in voordat de MSX het juiste antwoord geeft:

```
regel 40: MID$(A$,P,N)       is MID$(A$,4,8)  is ........
regel 50: MID$(A$,N,P)       is MID$(A$,8,4)  is ........
regel 60: MID$(A$,N+2,P-3)   is MID$(A$,10,1) is ........
regel 80: MID$(A$,P,N)       is MID$(A$,6,4)  is ........
```

92. Strings in stukjes

In alle opdrachten waarin we variabelen gebruiken kunnen we de waarde van die variabelen veranderen. Dat kan bijvoorbeeld met een FOR-NEXT opdracht. Dat geldt ook voor de twee getalvariabelen in de MID$ opdracht.

```
10 CLEAR 500
20 CLS
30 A$="WAAR ROOK IS IS VUUR"
40 FOR P=1 TO 20
50 PRINT MID$(A$,P,1)
60 NEXT P
70 STOP
```

Op regel 50 is de positie in de string A$ aangegeven met de variabele P. De waarde van deze variabele laten we variëren van 1 tot 20 in de FOR-NEXT op de regels 40, 50 en 60. Het resultaat dat we dus mogen verwachten ziet er zo uit:

P=1 dus MID$(A$,P,1) is MID$(A$,1,1) is W
P=2 dus MID$(A$,P,1) is MID$(A$,2,1) is A enzovoorts.

Alle letters komen één voor één aan de beurt van links naar rechts. De MSX stopt na de R van VUUR omdat hij dan alle 20 letters gehad heeft. Dat de MSX weet dat hij daar moet stoppen komt omdat wij van tevoren het aantal letters in de string op regel 30 geteld hebben. We kunnen natuurlijk ook de MSX laten bepalen hoeveel tekens er in de string zitten. We hoeven dan niet zelf te tellen en kunnen elke willekeurige string gebruiken. Verander regel 40 in:

```
40 FOR P=1 TO LEN(A$)
```

We kunnen de computer ook van achteren naar voren door de string laten werken. Om dat te bereiken veranderen we regel 40 zo:

```
40 FOR P=LEN(A$) TO 1 STEP -1
```

De MSX bepaalt eerst hoe lang de string A$ is en werkt dan terug van het eind naar het begin. Dus hier van positie 20 naar positie 1.

P=20 dus MID$(A$,P,1) is MID$(A$,20,1) is R
P=19 dus MID$(A$,P,1) is MIDS(A$,10,1) is U enzovoorts.

Als we regel 50 veranderen kunnen we de MSX de string A$ achterstevoren op het scherm laten zetten:

```
50 PRINT MID$(A$,P,1);          (; toegevoegd)
```

Verander regel 30 en 70 nog en dan hebben we een tekstomkeerprogramma gekregen:

```
30 INPUT "TEKST";A$
70 PRINT:GOTO 30
```

Het voorbeeld in deze paragraaf laat zien hoe we een tekst letter voor letter kunnen aflopen, van voor naar achter of, zo nodig, van achter naar voor.

We kunnen de MID$ ook in andere opdrachten gebruiken. Met een IF-opdracht kunnen we bijvoorbeeld een programma maken dat telt hoe vaak een bepaalde letter voorkomt in een tekst.

```
10 CLEAR 500
20 CLS
30 T=0
40 INPUT "TEKST";A$
50 FOR P=1 TO LEN(A$)
60 IF MID$(A$,P,1)="A" OR MID$(A$,P,1)="a" THEN T=T+1
70 NEXT P
80 PRINT "In deze tekst zitten"T"letters A of a"
90 PRINT
100 GOTO 30
```

We kunnen een tekst ook afzoeken naar een combinatie van letters als we het aantal tekens in de MID$ functie aanpassen. Als we bijvoorbeeld zoeken naar een combinatie van twee letters, de OE, in een tekst, moeten we steeds één positie verder gaan, maar kijken naar twee letters uit de string A$. Het zoeken verloopt dus zo:

```
A$="DE BOER ZOEKT ZIJN SNOEP OP DE STOEP"
```

P=	MID$(A$,P,2)=	
1	"DE"	
2	"E "	
3	" B"	
4	"BO"	
5	"OE"	
6	"ER"	enzovoorts.

In het volgende programma hebben we dit toegepast. Bij de INPUT-opdracht op regel 40 kunnen we typen naar welke letter, lettercombinatie of zelfs welk woord we gaan zoeken. De teksten die afgezocht worden staan op dataregels. We laten de MSX op regel 50 bepalen hoe lang de lettercombinatie is die we zoeken. Dit getal gebruiken we in de MID$ opdracht op regel 90.
Neem het programma over in de MSX en probeer het uit.

Opmerking: We hebben in dit programma alleen hoofdletters gebruikt. Als we ook kleine letters gebruiken wordt het zoeken lastiger.

```
10 CLEAR 500
20 CLS
30 T=0
40 INPUT "WAT ZOEKT U";A$
50 N=LEN(A$)
60 READ B$
70 IF B$="STOP" THEN STOP
80 FOR P=1 TO LEN(B$)
90 IF MID$(B$,P,N)=A$ THEN T=T+1
100 NEXT P
110 PRINT T "KEER " A$ " IN " B$
120 PRINT
130 GOTO 60
500 DATA "JANTJE ZAG EENS PRUIMEN HANGEN"
510 DATA "O ALS EIEREN ZO GROOT"
520 DATA "T SCHEEN DAT JANTJE WOU GAAN PLUKKEN"
530 DATA "SCHOON ZIJN VADER T HEM VERBOOD"
540 DATA STOP
```



In dit programma is iets niet in de haak! Hoe lossen we dat op?

In dit voorbeeld bepaalt de MSX op regel 50 de lengte van de string A$. De uitkomst komt in N te staan. Deze N wordt op regel 90 gebruikt in de opdracht MID$. Zo'n omweg is niet nodig. LEN(A$) stelt een getal voor. We kunnen dat "getal" ook direct in de MID$ opdracht gebruiken.
Regel 50 vervalt dan en regel 90 ziet er zo uit:

```
90 IF MID$(B$,P,LEN(A$))=A$ THEN T=T+1
```

Zulke combinaties zijn voor de MSX geen probleem. Alleen moeten wijzelf goed oppassen met het plaatsen van de haakjes en de komma's.

Opmerking: op de dataregels staan de teksten tussen aanhalingstekens. Dat mag, maar het is niet noodzakelijk. De MSX herkent de afzonderlijke "data" aan komma's of doordat ze op verschillende dataregels staan. Alleen als in een tekst een komma voorkomt (bijvoorbeeld als we van regel 510 willen maken "O, als eieren zo groot") zijn de aanhalingstekens nodig. De MSX ziet de komma dan niet als "scheidingsteken" tussen twee data, maar als deel van de tekst. We kennen dit verschijnsel al van de INPUT-opdracht.

Het volgende programma is een woordraadspelletje. De woorden die geraden moeten worden staan op dataregels. Elke keer leest de MSX een woord en vertelt ons met streepjes uit hoeveel letters dat woord bestaat. Daarvoor zorgen de regels 70, 80 en 90. Daarna kunnen we raden. We kunnen op regel 110 het hele woord typen of alleen één letter.
De MSX kijkt eerst of we het hele woord goed geraden hebben. Als dat het geval is krijgen we het volgende woord. Hebben we niet goed geraden, dan kijkt de MSX het woord dat geraden moet worden letter voor letter na en zoekt of de letter die wij geraden hebben in het woord staat.
Als de MSX de geraden letter tegenkomt op de positie die hij onderzoekt, zet hij de letter ook op het scherm. Staat op die positie de geraden letter niet, dan komt er een streepje op het scherm.

Een voorbeeld:

```
Te raden woord:  werkbank              --------

Wij raden: a
MSX geeft:                             -----a--
Wij raden: k
MSX geeft:                             ---k---k
```

Als we een nieuwe letter raden blijft de oude goed geraden letter dus niet staan. Neem het programma in de MSX over en probeer het uit.

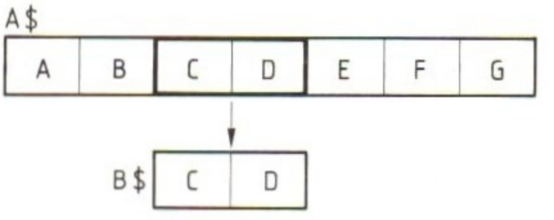
```
10 CLEAR 500
20 CLS
30 PRINT "Woorden raden"
40 READ A$
50 IF A$="STOP" THEN PRINT "einde van het spel":STOP
60 L=LEN(A$)
70 FOR I=1 TO L
80 PRINT "-";
90 NEXT I
100 PRINT
110 INPUT "Raad eens";B$
120 IF B$=A$ THEN PRINT "Geraden!":GOTO 40
130 FOR P=1 TO L
140 IF MID$(A$,P,1)=B$ THEN PRINT B$; ELSE PRINT "-";
150 NEXT P
160 PRINT
170 GOTO 110
500 DATA basic,telefoon,aktie,asbak,radio,stekker
510 DATA stopcontact,vooruitzicht,journaal,regering
520 DATA microcomputer,voorruit,spiegel,honderd
530 DATA STOP
```

Kies zelf een scherm (0 of 1) en de kleuren.

93. Een string veranderen

In deze paragraaf gaan we de functie MID$ nog op een andere manier gebruiken. Tot nu toe gebruikten we de MID$ om een stukje uit een string te halen en dat af te drukken of er een vergelijking mee te maken.
Nu gaan we de MID$ gebruiken om iets op een bepaalde plaats in een string te zetten.
Probeer het volgende programma uit:

```
10 CLS
20 A$="ABCDEFG"
30 B$=MID$(A$,3,2)
40 MID$(A$,3,2)="**"
50 PRINT B$
60 PRINT A$
70 STOP
```



In dit programma gebruiken we op regel 30 de opdracht MID$ op de "oude manier". We nemen een stukje A$ en zetten dat in B$. Vanaf positie 3 nemen we twee tekens, dat zijn dus de letters C en D, die naar B$ gaan. Door deze opdracht blijft de string A$ natuurlijk intact. Daar staat nog steeds ABCDEFG. Op regel 40 brengen we daar verandering in. We lichten nu niet een stukje uit A$, maar zeggen dat in A$ twee sterretjes geplaatst moeten worden en wel vanaf positie 3, twee tekens lang. Positie 3 en 4 worden dus veranderd in sterretjes.
Het resultaat van deze twee opdrachten blijkt op regel 50 en 60. Dat biedt weer interessante mogelijkheden die we onderzoeken met het volgende programma:

```
10 CLS
20 CLEAR 500
30 A$="ABCDEFG"
40 B$="1234567"
50
60 PRINT A$"_"B$          ( _ = spatie)
70 STOP
```

Regel 50 van dit programma is open gelaten. Hier volgen een paar mogelijkheden om deze regel in te vullen.

a. MID$(A$,4,1)="*"
b. MID$(B$,1,5)="*****"
c. MID$(A$,1,4)="*"
d. MID$(B$,2,1)="AB"
e. MID$(A$,2,1)=MID$(B$,2,1)
f. MID$(A$,2,1)=MID$(A$,3,1)

De conclusie na dit experiment is niet moeilijk: met de functie MID$ kunnen we iets uit een string halen of er iets in zetten. Waar we dat vandaan halen mogen we helemaal zelf weten. Het is zelfs mogelijk om iets op een bepaalde plaats in een string te zetten dat van een andere plaats uit dezelfde string komt. Dat blijkt bij f hierboven waar we het teken van positie 3 brengen naar positie 2 van de string A$.
Merk op wat de MSX doet als het aantal tekens niet klopt zoals bijvoorbeeld bij c hierboven.

Met deze nieuwe wetenschap kunnen we ons woordraadspel verbeteren:

```
10 REM**WOORDRADEN**
20 CLEAR 500
30 CLS
40 PRINT "Woorden raden"
50 PRINT
60 READ W$:REM lees het te raden woord
70 IF W$="STOP" THEN PRINT "Einde van het spel":STOP
80 S$="":REM S$ begint als "lege string"
90 FOR I=1 TO LEN(W$)
100 S$=S$+"-":REM S$ wordt evenveel streepjes als W$ lang is
110 NEXT I
120 PRINT S$
130 INPUT "Raad eens";R$:REM R$ is het geraden woord of de geraden letter
140 IF R$=W$ THEN PRINT "Geraden!":GOTO 50
150 FOR P=1 TO LEN(W$):REM zoek W$ af
160 IF MID$(W$,P,1)=R$ THEN MID$(S$,P,1)=R$:REM als de letter "klopt" zetten
    we hem op de juiste plaats in S$, anders laten we het streepje staan
170 NEXT P
180 GOTO 120
```

Voeg aan dit programma een rij woorden toe op dataregels (het laatste woord moet in hoofdletters STOP zijn) en schrijf daarna het programma op de cassette met de opdracht CSAVE. Geef zelf een naam aan dit programma.

94. Getal of Tekst?

In deze paragraaf nog drie nieuwe opdrachten over strings.

STRING$(aantal,teken)

Met deze opdracht kunnen we een string maken van een aantal gelijke tekens. Het teken zelf kan in de opdracht staan, natuurlijk tussen aanhalingstekens, maar we mogen ook een stringvariabele of -functie vermelden.
Voorbeelden: stel Q$="ABCDEF"

```
a.  A$=STRING$(10,"*")            dus A$ wordt "**********"
b.  B$=STRING$(10,Q$)             dus B$ wordt "AAAAAAAAAA"
c.  C$=STRING$(10,MID$(Q$,5,1))   dus C$ wordt "EEEEEEEEEE"
d.  D$=STRING$(10,CHR$(61))       dus D$ wordt "=========="
```

In voorbeeld b zien we dat van de string Q$ alleen het eerste teken genomen wordt. Dat komt 10 keer in B$ te staan.

Op de plaats van het aantal in de opdracht kan een getal of een variabele of een berekening staan.
Op regel 100 in het woordraadsel hebben we een string van streepjes gemaakt die even lang is als een bestaande string. Met de STRING$ opdracht kunnen we dat eenvoudiger doen:

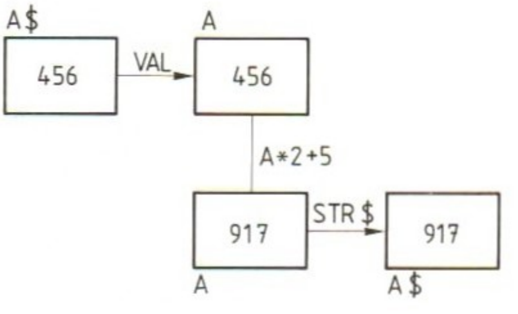
```
100 L=LEN(W$):S$=STRING$(L,"-")  of, nog korter,
100 S$=STRING$(LEN(W$),"-")
```

De opdracht PRINT is een van de weinige opdrachten die we zowel voor strings als voor getallen kunnen gebruiken. De meeste opdrachten werken alleen maar voor strings of alleen voor getallen. Daarom is het soms nodig een string om te zetten in een getal of, omgekeerd, een getal om te zetten in een string. Daarvoor hebben we twee opdrachten:

VAL(string) maakt van een string een getal om mee te rekenen
STR$(getal) maakt van een getal een string (om b.v. MID$ te kunnen gebruiken)

Het volgende programma laat zien wat de opdrachten VAL en STR$ doen.

```
10 CLEAR 500
20 CLS
30 INPUT "Geef string";A$
40 PRINT "String is:"A$
50 A=VAL(A$)
60 PRINT "Getal is:"A
70 A=A*2+5
80 PRINT "Getal wordt:"A
90 A$=STR$(A)
100 PRINT "String wordt:"A$
110 PRINT
120 GOTO 30
```



Neem dit programma over in de MSX, controleer of het goed werkt en ga daarna met de volgende opdrachten na wat de opdrachten VAL en STR$ precies doen.

	String (regel 30-40)	Getal (regel 60)	Getal (regel 80)	String (regel 100)
a.	40	...	...	...
b.	-40	...	...	...
c.	+40	...	...	...
d.	40+	...	...	...
e.	40-	...	...	...
f.	4.7	...	...	...
g.	-4.7	...	...	...
h.	+4.7	...	...	...
i.	4,7	...	...	...
j.	7E3	...	...	...
k.	-7E3	...	...	...
l.	JAN	...	...	...
m.	"4,7"	...	...	...
n.	6%	...	...	...
o.	3/4	...	...	...
p.	1.000.000	...	...	...

95. Invoeren met INPUT$ en INKEY$

De INPUT-opdracht die we uit vorige hoofdstukken al kennen, heeft twee nadelen:

1. We kunnen zoveel invoeren als we willen en ook toetsen die ons scherm vernielen zodat er niet meer te zien is wat we nu eigenlijk moeten doen.
2. We moeten een ingevoerd getal of een ingevoerde tekst altijd afsluiten met de RETURN-toets, ook als we maar één teken hoeven te typen.

De MSX heeft drie andere invoeropdrachten waarmee we een deel van deze problemen kunnen omzeilen. Dat zijn:

LINE INPUT (die later besproken wordt)
INPUT$(...) (een functie)
INKEY$

Met de opdracht INPUT$(...) kunnen we de MSX om een beperkt aantal tekens laten vragen. Tussen de haakjes komt het aantal tekens te staan. Tijdens het invoeren zijn de getypte tekens niet te zien.

Probeer:

```
10 CLEAR 500
20 X$=INPUT$(10)
30 PRINT X$
40 PRINT
50 GOTO 10
```

Probeer het programma uit en kijk wat er gebeurt. Na de opdracht RUN laat de MSX géén vraagteken zien. We zien dus niet dat er iets ingevoerd moet worden. We zien dat we persé 10 tekens moeten invoeren, maar dat mogen alle toetsen zijn. Probeer bijvoorbeeld maar 10 keer HOME of 10 keer BS.

Als we willen verbeteren met BS geeft dat problemen. Als we bijvoorbeeld invoeren:

aaa BS BS BS (om de drie a's weer weg te halen)

hebben we al 6 tekens ingevoerd. We kunnen dus nog maar 4 tekens typen, bijvoorbeeld:

bbbb

Zonder op RETURN te drukken gaat de MSX nu weer verder met het programma.

Als we de opdracht gebruiken in de vorm INPUT$(1) laten we de MSX vragen om het invoeren van één teken. We kunnen, nadat dit teken ingevoerd is, kijken of het aan onze wensen voldoet. Erg handig is daarbij de opdracht INSTR.

Typ het volgende programma:

```
10 CLEAR 500
20 CLS
30 PRINT "Gaan we beginnen?";
40 X$=INPUT$(1)
50 IF INSTR("jJnN",X$)=0 THEN 40
60 PRINT "Je typte een " X$
70 IF INSTR("jJ,X$)=0 THEN 30
80 PRINT "We beginnen!"
```

Regel 10 zorgt voor de vraag. De puntkomma zorgt ervoor dat de volgende opdracht op dezelfde regel op het scherm wordt uitgevoerd.
Als de MSX regel 40 uitvoert wacht hij tot er één toets is getypt. Zodra dat is gebeurd gaat hij verder met regel 50. Daar staat de nieuwe functie INSTR.

De functie INSTR levert een getal op. De MSX kijkt op welke plaats (positie) de tweede string (hier X$, dat ene, ingevoerde teken) voorkomt in de eerste string (hier de kleine en grote j en n, "jJnN").

Voorbeeld:
P=INSTR("abcdefg","b") dan wordt P=2
Q=INSTR("abcdefg","d") dan wordt P=4
M=INSTR("abcdefg","h") dan wordt P=0

Als de functie INSTR een getal 0 oplevert, komt de gezochte string niet in de eerste string voor. De twee strings mogen zelf in de functie staan, maar ook als variabele. Voorbeeld:

A$="abcde":B$="c"
P=INSTR(A$,BS) dan wordt P=3

In ons programma kijken we dus op regel 50 of het ingevoerde teken in X$ ergens in de string "jJnN" voorkomt. Als dat <u>niet</u> zo is (de uitkomst van INSTR is dan 0) gaan we terug naar regel 40. Er moet opnieuw ingevoerd worden. We zorgen er daarmee voor dat we alleen verder gaan met het programma als één van de 4 tekens j, J, n of N getypt zijn. Omdat we niets printen voordat we zeker weten dat alleen één van deze vier gekozen is, kan er ook met het scherm niets verkeerd gaan. Probeer de toetsen CLR, BS e.d. maar eens: de MSX reageert er niet op.

Op regel 60 zetten we op het scherm wat er gekozen is. Kijk eens waar dat op het scherm terecht komt.

Op regel 70 gebruiken we de INSTR nog een keer, maar nu kijken we of het getypte teken een j of een J is (de n en de N vallen dus af).

Dat had ook zo gekund:

```
70 IF X$<>"j" AND X$<>"J" THEN 30
```

Dat betekent voor de MSX: als X$ geen "j" is en ook geen "J", dan gaan we naar regel 30 (en beginnen dus niet).

Opmerkingen:

1. De toets CTRL-STOP werkt nog wel bij regel 40, maar CTRL-C niet!
2. De opdracht INPUT$(1) is heel geschikt voor dit werk. De opdracht INPUT$(...) met een getal groter dan 1 tussen de haakjes is eigenlijk niet geschikt voor invoer op het toetsenbord. We kunnen hem wel goed gebruiken bij het invoeren (lezen) van gegevens uit een bestand op bijvoorbeeld cassette.

De tweede opdracht die we kunnen gebruiken is de INKEY$. De $ laat zien dat het ook bij deze opdracht om een string gaat. De INKEY werkt anders dan de INPUT of INPUT$. De INKEY$-opdracht laat de MSX niet wachten tot er een toets ingedrukt is, maar kijkt even óf er een toets getypt is en gaat meteen verder met de volgende opdracht.
Probeer maar:

```
10 CLEAR 500
20 CLS
30 PRINT "Gaan we beginnen?";
40 X$=INKEY$
50 PRINT X$
60 STOP
```

Het resultaat van een RUN-opdracht is:

```
Gaan we beginnen?
Break in 50
Ok
```

De MSX wacht niet, zoals hij normaal bij een INPUT-opdracht doet. We moeten dus de MSX "tegenhouden" bij de INKEY$-opdracht. Dat kan zo:

```
40 X$=INKEY$:IF X$="" THEN 40
```

Probeer het programma met deze toevoeging opnieuw. Het programma blijft na de tekst "Gaan we beginnen?" netjes wachten: zolang wij niet op een toets hebben gedrukt blijft de string X$ "leeg" (dus gelijk aan "") en wachten we door de GOTO naar dezelfde INKEY$-opdracht.

Maak nu het programma af:

```
50 IF INSTR("jJnN",X$)=0 THEN 40
60 PRINT "Je typte een " X$
70 IF INSTR("jJ,X$)=0 THEN 30
80 PRINT "We beginnen!"
```

Opmerking: let op! Als we niet op één of andere manier nagaan wat er ingevoerd is voordat we dat met PRINT op het scherm zetten, wordt alles als invoer geaccepteerd, dus ook bijvoorbeeld HOME, CLR, BS e.d. Dat is alleen bruikbaar in een situatie waarin we alleen maar willen dat er één of andere toets wordt aangeslagen, voordat het programma verder gaat. We kunnen daarmee bijvoorbeeld "leestijd" geven: het scherm staat vol informatie, we zetten onderaan op het scherm:

Druk op een toets om verder te gaan

en programmeren:

```
120 X$=INKEY$:IF="" THEN 120
```

(Het regelnummer is maar een voorbeeld).

Probeer:

```
10 H$=""
20 L$="abcdefghijklmnopqrstuvwxyz"
30 PRINT "Typ een woord van vijf letters: ";
40 FOR P=1 TO 5
50 X$=INPUT$(1)
60 IF INSTR(L$,X$)=0 THEN 50
70 H$=H$+X$
80 PRINT X$;
90 NEXT P
100 PRINT
110 PRINT "Het woord is "H$
120 GOTO 10
```

H$ begint als "lege" string. We mogen een woord typen van vijf kleine letters. Alle andere tekens worden geweigerd (behalve de CTRL-STOP-toets!). De getypte letter komt op het scherm en wordt aan H$ vastgeplakt.

Vraag: waarvoor dient regel 100?

96. Opdrachten

Opdracht 1:
Maak het woordraadspel mooier door het gebruik van kleuren.
Voeg opdrachten toe, die tellen hoeveel beurten gebruikt worden om het woord te raden, zodat we op het scherm te zien krijgen (bijvoorbeeld!): Geraden in 6 beurten.

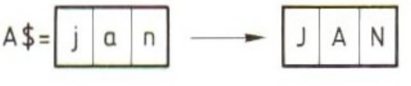
Opdracht 2:
De opdracht MID$ kan op drie manieren gebruikt worden:

```
A$=MID$(B$,P,N)
MID$(A$,P,N)=B$
MID$(A$,P,N)=MID$(B$,X,Y)
```

De gebruikte variabelen zijn maar voorbeelden.

Onderzoek of de opdrachten LEFT$(A$,N) en RIGHT$(A$,N) ook op deze drie manieren gebruikt kunnen worden.

Opdracht 3:
Een tekst, die bij een INPUT-opdracht getypt wordt, kan hoofd- en kleine letters bevatten. Maak een stukje programma, dat alle kleine letters in de string A$ omzet in hoofdletters.
Let op: cijfers en leestekens moeten niet omgezet worden. Het gaat alleen om de <u>letters.</u>

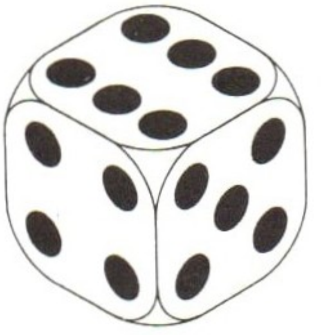


HOOFDSTUK 12: In de rij; tabellen

97. Tabellen

Tabellen (rijen, Engels: arrays) zijn erg handig en worden veel gebruikt. In deze paragraaf laten we eerst zien welke soort problemen we met een tabel kunnen oplossen. We laten de MSX duizend keer met een dobbelsteen gooien en tellen hoe vaak hij 1, 2, 3, 4, 5 of 6 gooit.

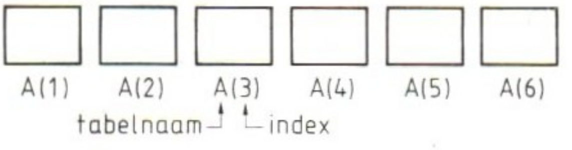
```
10 CLS
20 FOR I=1 TO 1000
30 A=INT(RND(5)*6)+1
40 IF A=1 THEN A1=A1+1:GOTO 100
50 IF A=2 THEN A2=A2+1:GOTO 100
60 IF A=3 THEN A3=A3+1:GOTO 100
70 IF A=4 THEN A4=A4+1:GOTO 100
80 IF A=5 THEN A5=A5+1:GOTO 100
90 A6=A6+1
100 NEXT
110 PRINT "1:" A1 "keer"
120 PRINT "2:" A2 "keer"
130 PRINT "3:" A3 "keer"
140 PRINT "4:" A4 "keer"
150 PRINT "5:" A5 "keer"
160 PRINT "6:" A6 "keer"
170 STOP
```



Geen moeilijk programma, maar wel lang. Waarom hebben we in dit programma zoveel opdrachten nodig?

Dat komt omdat we zes verschillende dingen willen tellen. Daarvoor hebben we in het geheugen van de MSX zes variabelen nodig.
Om nu uit te zoeken in welk van die zes variabelen we moeten tellen moeten we al die vergelijkingen maken. De variabelen hebben verschillende namen, dus hebben we verschillende PRINT-opdrachten nodig om de resultaten op het scherm te krijgen. Bij het tellen van de ogen van de dobbelsteen is dat allemaal nog best te doen. Het is geen groot programma.
Vervelend wordt het wel als we de MSX niet met de dobbelsteen willen laten gooien maar bijvoorbeeld de trekking van de lotto laten doen. Dan moeten we tellen hoe vaak elk van de getallen van 1 t/m 41 voorkomen. Dat zou dus betekenen: 40 vergelijkingen en 41 PRINT-opdrachten.

Het kan wel anders, maar dan moeten we gebruikmaken van een tabel of rij. Wat is het bijzondere van een tabel? Een tabel is een rij vakjes in het geheugen met dezelfde naam.
Om het telprobleem op te lossen gebruiken we een tabel van 6 vakjes. Die tabel geven we een naam, bijvoorbeeld A. Alle zes vakjes in de tabel heten nu A.
Zo zonder meer kunnen we die vakjes niet gebruiken. We moeten de MSX duidelijk maken welke van de 6 vakjes A we bedoelen in een opdracht. Dat doen we door de vakjes naast een naam ook nog een nummer te geven. We nummeren de vakjes van 1 t/m 6. Het nummer van het vakje schrijven we tussen haakjes achter de naam.
Het ziet er dan zo uit:



Een vakje in een tabel heeft dus een naam (de naam van de tabel) en een nummer. Dit nummer, dat altijd tussen haakjes staat, noemen we de index. Elke keer als een getal in een opdracht ter sprake komt, weten we dat op de plaats van dat getal ook een variabele kan staan. Dat geldt ook voor de index. Op de plaats van de index mag een variabele staan, maar ook een berekening met getallen en variabelen.

Opdracht 1:
Vul in:

a. A(5) is tabel A , vakje 5
b. B(12) is tabel ..., vakje ...
c. C(24) is tabel ..., vakje ...
d. P$(3) is tabel ..., vakje ...

e. X=5, A(X) is dan tabel A , vakje 5
f. X=7, B(X) is dan tabel ..., vakje ...
g. Q=12, C(Q) is dan tabel ..., vakje ...
h. P=3, P$(P) is dan tabel ..., vakje ...

i. X=4, A(X+1) is dan tabel A , vakje 5
j. Z=7, B(Z+2) is dan tabel ..., vakje ...
k. Q=12 en G=4, C(Q+G) is dan tabel ..., vakje ...
l. Q=12 en G=4, P$(Q-G) is dan tabel ..., vakje ...

98. Het gemak van de functietoetsen

In de volgende paragrafen moeten we nogal eens hetzelfde typen. We kunnen ons dat gemakkelijker maken door de edit-mogelijkheden van de MSX te gebruiken, maar er is nog een ander handig hulpmiddel: de functietoets.

Tot hier hebben we de functietoetsen gebruikt met de betekenissen die ze "vanzelf" hebben. Die betekenissen staan op de laatste schermregel:

zonder SHIFT:	F1	F2	F3	F4	F5
	color	auto	goto	list	run
	color 15,4,7	cload"	cont	list.	run
met SHIFT :	F6	F7	F8	F9	F10

Deze betekenissen kunnen we ook op een andere manier zichtbaar maken, namelijk met:

Wij: **key list**
MSX: laat de betekenissen onder elkaar zien

We kunnen de betekenis-regel van de functietoetsen ook van het scherm weghalen, namelijk met de opdracht:

Wij: **key off**
MSX: maakt regel 24 schoon

Deze 24-ste regel van het scherm kunnen we nu gewoon gebruiken voor ons werk. Het scherm is dus een regel groter geworden.

Wij: **key on**
MSX: laat de regel met functietoetsen weer zien

Het aardige van de functietoetsen is, dat wij ook zelf een betekenis aan zo'n toets kunnen geven. Dat doen we met de opdracht KEY. In deze opdracht moet natuurlijk staan welke toets we bedoelen en wat de betekenis wordt.
Probeer dit voorbeeld:

Wij: **key 1,"A$"**

Kijk nu eens op de laatste schermregel. Daar staat de nieuwe betekenis van toets F1. Als we op F1 drukken komt er op het scherm: A$. Probeer het maar!

We kunnen nu aan elke functietoets een betekenis geven van een opdracht die we vaak gebruiken. Dat scheelt veel typewerk!

Probeer zelf:

key 1,"CLS"
key 6,"INPUT$(1)"
key 7,"A( )" (met een spatie tussen de twee haakjes)
key 8,"A$( )" (met een spatie tussen de twee haakjes)

Probeer ook:

key 1
key 1,
key 1,CLS

en ook:

A$="PRINT"
key 1,A$

De opdracht key hangt een string (in de vorm van een echte string of in de vorm van een variabele) aan de gekozen functietoets. Als we op de functietoets drukken voeren we de tekst, die bij deze toets hoort, in. Dat gebeurt tijdens het programmeren, maar ook bij INPUT-opdrachten. Maar, nogmaals: het gaat altijd om strings, nooit om getallen!
De opdracht KEY neemt nooit meer dan 15 tekens op.

In de tekst die we aan een functietoets hangen, kunnen we ook stuurcodes verstoppen. Omdat we die niet met één toets kunnen typen, moeten we ze aan de tekst vastplakken. Probeer dit maar eens:

key 4,CHR$(12)+"list"

en probeer dan F4. Wat gebeurt er? De MSX maakt eerst het scherm schoon (omdat hij de CHR$(12) op het scherm zet) en laat dan het woord list zien.
Direct achter dit woord blijft de cursor staan.
Dat kunnen we ook nog handiger maken (geef eerst even RETURN):

key 1,CHR$(12)+"list"+CR$(13)

Sla nu F4 maar eens aan! De CHR$(13) zorgt ervoor dat we niet eens meer RETURN hoeven te typen. De list komt direct op een schoon scherm.

Kijk nu wat de betekenis is van de toetsen met KEY LIST. De betekenis die we aan toets F7 hebben gegeven, kunnen we in de volgende paragrafen goed gebruiken. Laat die in elk geval staan.

99. De inhoud van een tabel-element

In deze paragraaf gaan we kijken wat er in de vakjes van een tabel staat. Dus niet naar de variabele, maar naar de waarde die bij die variabele hoort. We gaan uit van twee tabellen. Dit zijn ze:

A(1)=5	A(2)=19	A(3)=20	A(4)=6
P$(1)="JAN"	P$(2)="HENK"	P$(3)="TRUDY"	P$(4)="WILMA"

In vakje 2 van tabel A staat de waarde 19. In vakje 3 van tabel B$ staat de waarde (string) TRUDY. Met PRINT-opdrachten kunnen we op het scherm krijgen wat er in een vakje staat:

Vul in:

Opdrachten	Op het scherm
a. PRINT A(1)	5
b. PRINT P$(1)	JAN
c. PRINT A(3)	...
d. PRINT P$(4)	...
e. PRINT ...(...)	TRUDY
f. PRINT ...(...)	6

We maken nu gebruik van de mogelijkheid om als index een variabele of een berekening te gebruiken.

Vul in:

Opdrachten	Op het scherm
g. X=4:PRINT A(X)	6
h. X=5:PRINT P$(X-1)	...
i. X=7:PRINT A(A-5)	...
j. X=2:Y=1:PRINT P$(X+Y)	...
k. X=2:Y=1:PRINT A(X-Y)	...

Een vakje in een tabel noemen we een tabel-element. In de voorbeelden in deze paragraaf hebben we twee tabellen. Tabel A met vier elementen, genummerd van 1 t/m 4 en tabel P$, ook met vier elementen, genummerd van 1 t/m 4.

We gaan nu kijken of de MSX ook met tabellen en tabel-elementen overweg kan. We proberen dat eerst uit met een paar losse opdrachten. Maak het geheugen van de MSX leeg en probeer daarna:

Wij : **A(1)=10** (Het mag ook op één regel, maar dan met : ertussen!)
A(2)=20
A(3)=30
A(4)=40

De A ( ) kunnen we typen met de functietoets F7. We kunnen dan de cursor twee plaatsen terugzetten om het getal tussen de haakjes in te vullen. We kunnen het ons zelfs nog gemakkelijker maken door dat terugzetten van de cursor alvast door de MSX te laten doen. In een vorig hoofdstuk hebben we al gezien dat dat kan met de CHR$(29):

key 7,"A( )"+CHR$(29)+CHR$(29)

De cursor komt dan mooi tussen de haakjes te staan. We kunnen A(6) dan gemakkelijk zo typen:

Wij: **F7**
MSX: A(_)   (_ hier staat de cursor)
Wij: **6**

Op deze opdrachten reageert de MSX alleen met Ok. Wat we gevraagd hebben heeft hij wel gedaan. In tabel-element A(1) heeft hij de waarde 10 gezet. In tabel-element A(4) de waarde 40.
Dat die waarden erin zitten kunnen we zien met de volgende opdrachten:

Wij: **PRINT A(1);A(4)**
MSX: **10 40**

We gaan nu rekenen met de waarden in de tabel-elementen. Probeer:

Wij: **PRINT A(3)+A(4)**
MSX: **70**
Wij: **A(5)=A(1)+A(2):PRINT A(5)**
MSX: **30**

Eerst hebben we de som van de elementen 3 en 4 op het scherm gezet. Daarna hebben we in element 5 van tabel A een nieuwe waarde gezet. Die waarde heeft de MSX berekend: 10 + 20 = 30. Deze 30 komt in A(5).
De tabel in het geheugen van de MSX ziet er nu dus zo uit:

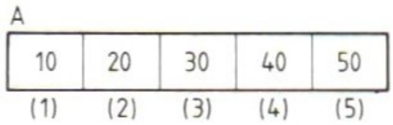
A(1)=10   A(2)=20   A(3)=30   A(4)=40   A(5)=30

We gaan nu rekenen met de <u>index</u>. Probeer:

Wij: **PRINT A(3)+1**   (+1 buiten de haakjes)
MSX: **31**
Wij: **PRINT A(3+1)**   (+1 binnen de haakjes)
MSX: **40**
Wij: **PRINT A(3+1)+1**   (+1 binnen en buiten de haakjes)
MSX: **41**

Eerst hebben we de MSX opdracht gegeven bij de waarde in element 3, dat is dus het getal 30, 1 op te tellen en dat op het scherm te zetten. Het antwoord van de MSX is dus 31.
Daarna kreeg de MSX opdracht de waarde in element 3+1, dat is dus element 4, op het scherm te zetten. Die waarde 40 is verschenen.
Tenslotte hebben we die beide opdrachten gecombineerd. De MSX moest de waarde in element 3+1, dus element 4, bepalen en daarbij 1 optellen.
De uitkomst, 41, kwam op het scherm. Probeer nu zelf de volgende opdrachten uit:

```
PRINT A(5)
PRINT A(5)+5
PRINT A(5)+A(1)
PRINT A(1)+A(3)+A(4)
PRINT A(0)
PRINT A(7)
PRINT A(2)+3
PRINT A(2)+A(3)
PRINT A(2+3)
```



Als we de MSX opdracht geven element 0 op het scherm te zetten protesteert hij niet. Op het scherm verschijnt het getal 0. Dat gebeurt ook als we de MSX naar element 7 vragen. De MSX protesteert niet. Dat kan alleen betekenen dat hij deze twee elementen van de tabel kent. Omdat wij daar geen getal in gezet hebben komt er 0 op het scherm.
We kunnen ons nu afvragen hoe ver we hiermee kunnen gaan.

Wij: **PRINT A(-1)**
MSX: **Illegal function call**

Een negatieve index, dat wil dus zeggen een negatief getal tussen de haakjes, is verboden.

Wij: **PRINT A(10)**
MSX: **0**
Wij: **PRINT (11)**
MSX: **Subscript out of range**

De MSX reageert met een nieuw bericht. Dit bericht betekent dat we te ver zijn gegaan. We gaan met onze index (Engels: subscript) buiten ons boekje. Voordat we conclusies gaan trekken kunnen we ook de volgende opdrachten nog proberen:

PRINT A(1.4) (gebruik . in plaats van ,)
PRINT A(1.6)

Conclusies:

a. zo zonder meer bestaat een tabel uit 11 elementen. Die elementen zijn genummerd van 0 t/m 10;
b. als we een te hoge index gebruiken krijgen we het bericht Subscript out of range;
c. als we een negatieve index gebruiken krijgen we het bericht Illegal function call;
d. als we een getal met cijfers achter de komma als index gebruiken rondt de MSX af op de gewone manier.

Opmerking: Negatieve getallen en getallen met cijfers achter de komma zouden kunnen ontstaan als uitkomst van een berekening. Als zo'n uitkomst negatief wordt zitten we fout. Als die uitkomst cijfers achter de komma heeft rondt de MSX af.

100. DIMmen

De MSX kent een opdracht waarmee wij de computer kunnen vertellen hoe groot een tabel moet zijn, d.w.z. hoeveel elementen er in de tabel moeten zitten. Als we die opdracht niet gebruiken neemt de MSX aan dat het er 11 moeten zijn, genummerd van 0 t/m 10. Deze opdracht is:

DIM naam(getal) (DIM van DIMension)

Voor de naam van een tabel gelden de gewone regels die we al kennen van de variabelen. We mogen de naam zo groot maken als we willen, maar alleen de eerste twee letters tellen. De naam moet met een letter beginnen. Een tabel waarin we strings willen opbergen krijgt een naam van ten minste één letter met daarachter een dollarteken. Het getal tussen de haakjes geeft aan wat de grootste index is die we willen gebruiken.

Voorbeelden:
DIM B(4) geeft B(0), B(1), B(2), B(3), en B(4)
DIM Q$(8) geeft Q$(0) t/m Q$(8), dus 9 elementen
DIM Q$(8),B(4),C(100),F(4),AN$(7)

Opmerking: als we deze voorbeelden zo uitproberen zoals ze hier staan, krijgen we op de derde regel problemen. De MSX meldt "Redimensioned array". Dat wil zoveel zeggen als: ik heb al een tabel (array) met deze naam! Ik kan er niet nog een maken.

Met deze laatste opdracht hebben we vijf tabellen opgegeven met respectievelijk 9, 5, 101, 5 en 8 elementen. We kijken nu even verder naar tabel B uit deze opdracht:

B(0)=0 B(1)=0 B(2)=0 B(3)=0 B(4)=0

In deze tabel zitten vijf elementen genummerd van 0 t/m 4. Element 5 bestaat in deze tabel niet. Dat komt door de DIM-opdracht waarin staat dat B genummerd wordt van 0 t/m 4. Alleen als we voor een tabel geen DIM-opdracht gebruiken krijgt die tabel automatisch 11 elementen genummerd van 0 t/m 10. In de tabel-elementen zet de MSX automatisch het getal 0.

Probeer nu de volgende opdrachten:

```
Wij: DIM B(4), C(30)
     PRINT B(-1)
     PRINT B(0)
     PRINT B(4)
     PRINT B(5)
     PRINT A(5)
     PRINT A(11)
     PRINT C(11)
```

Deze opdrachten laten zien dat we in de DIM-opdracht bepalen tot hoe ver we met de index kunnen gaan. Een negatieve index kan nooit. De tabel A komt in de DIM-opdracht niet voor. Die krijgt dus automatisch 11 elementen met als hoogste index 10.

Twee opmerkingen:

a. de opdracht DIM komt meestal vooraan in het programma te staan, net als de opdracht CLEAR. Deze twee opdrachten hebben ongeveer dezelfde functie. De DIM zorgt voor ruimte in het geheugen voor tabellen, de CLEAR voor ruimte voor strings. **Pas op: de CLEAR moet vóór de DIM, omdat CLEAR de werking van DIM weer opheft!!**
b. We mogen net zo veel DIM-opdrachten gebruiken als we willen. Ze mogen ook op verschillende regels staan. Wat niet mag is twee DIM-opdrachten voor één en dezelfde tabel geven.

Dit mag dus wel:

```
10 DIM B(5)
20 DIM B$(30)
30 DIM BA(17)
```

Dit mag dus niet:

```
10 DIM B(5)
20 DIM B$(30)
30 DIM B(17)
```

Laat de MSX maar vertellen welk bericht hij in dit geval geeft.

101. Werken met tabellen

Op zich maakt het niets uit of we een vakje in het geheugen nu A1 noemen of A(1). In het eerste geval hebben we een gewone variabele, in het tweede een variabele in een tabel. In beide gevallen zetten we gewoon in de opdracht welk vakje in het geheugen we bedoelen. Bijvoorbeeld:

```
PRINT A1
PRINT A(1)
```

In beide opdrachten staat heel concreet welke variabele op het scherm gezet moet worden. Daar kunnen we niets aan veranderen of we moeten het programma wijzigen. De tabel wordt pas interessant als we geen index in de vorm van een getal gebruiken, maar een index in de vorm van een variabele. Als we de volgende opdracht geven:

```
PRINT A(X)
```

weet niemand precies welk vakje in het geheugen we nu op het scherm gezet willen hebben. Dat hangt af van het getal dat **op een zeker moment** in de variabele X staat. Om een getal in X te krijgen kunnen we gebruikmaken van alle mogelijkheden die we tot nu toe hebben leren kennen. We kunnen het getal voor X typen op het toetsenbord, berekenen of uit een string halen. Maar wat nog belangrijker is: we kunnen de MSX die X bij het uitvoeren van het programma laten veranderen. Als we dat doen staat er de ene keer in deze PRINT-opdracht bijvoorbeeld 'zet het derde element van tabel A op het scherm' en een volgende keer 'zet het vierde element van tabel A op het scherm'.
Deze mogelijkheden gaan we uitproberen. We werken met twee kleine tabellen:

```
10 CLEAR 500
20 DIM A(4),A$(4)
```

De opdracht CLEAR geeft ons ruimte voor strings, die we ook voor de stringtabel nodig hebben. De DIM-opdracht maakt twee tabellen van vijf elementen, allebei genummerd van 0 t/m 4. Vervolgens gaan we in deze tabellen waarden zetten:

```
30 A(1)=10:A(2)=20:A(3)=30:A(4)=40                (denk aan de : !)
40 A$(1)="A":A$(2)="B":A$(3)="C":A$(4)="D"
50 CLS
```

In tabel A staan nu de getallen 10 t/m 40, in de tabel A$ de letters A t/m D. A(0) en A$(0) hebben geen waarde gekregen. A(0) heeft daardoor de waarde nul, A$(0) is een "lege" string.
Door de regels 30 en 40 hebben de elementen in de tabel nu een inhoud gekregen. We gaan daarmee aan het werk. We beginnen willekeurig uit de twee tabellen getallen en letters op te vragen. De index typen we op het toetsenbord. Voeg toe:

```
60 INPUT "INDEX";X
70 PRINT X;A(X);A$(X)
80 GOTO 60
```

De bedoeling van dit programma is dat wij een index typen. De MSX zoekt vervolgens op welk getal en welke letter bij deze index behoort en zet die op het scherm samen met de index zelf. Neem het hele programma over in het geheugen van de MSX en probeer het daarna uit met de volgende getallen:

Typ voor X	Op het scherm komt		
1	1	10	A
2	...	...	...
3	...	...	...
4	...	...	...
1.4	...	...	...
1.5	...	...	...
5	...	...	...

Bij de laatste regel van dit voorbeeld gaat het fout. Op het scherm komt:

Subscript out of range in 70

We zijn buiten ons boekje gegaan. Bij dit soort programma's is het altijd nodig ervoor te zorgen dat de index niet negatief kan worden en niet te groot wordt. In dit programma kunnen we dat bijvoorbeeld met de volgende opdracht oplossen:

65 IF X > 4 OR X < 1 THEN 60

We hebben er nu voor gezorgd dat de MSX niet met een index groter dan 4 in de tabellen kan gaan zoeken. Ook met een index kleiner dan 1 gaat hij niet aan het werk, maar vraagt op regel 60 om een nieuwe index. We hebben met opzet X kleiner dan 1 in regel 65 gezet. Dat betekent dat X = 0 ook niet meedoet. Dat hebben we gedaan omdat we in deze tabellen alleen de elementen 1 t/m 4 willen gebruiken. Element 0 bestaat wel, maar we gebruiken hem niet. We mogen element 0 wel gebruiken maar we zullen zien dat het vaak erg handig is om dat niet te doen en gewoon met 1 te beginnen.
Voeg nu regel 65 aan het programma toe en probeer het nog een keer uit.

We hebben gezien dat we uit de tabel een element kunnen lichten en op het scherm kunnen zetten of kunnen gebruiken in bijvoorbeeld een berekening. We halen nu de regels 60 t/m 80 van het programma af en gaan iets anders proberen. Regels 10 t/m 50 moeten dus blijven staan, anders moeten we de twee tabellen weer helemaal opnieuw intypen.
Geef een opdracht:

DELETE 60-80

Voeg daarna de volgende regels aan het programma toe:

```
60 X=1
70 PRINT X;A(X);A$(X)
80 X=X+1
90 GOTO 70
```

Dit programma gaat de twee **tabellen op het rijtje af langs** en zet alle elementen netjes op hun beurt onder elkaar op het scherm. Ook dit programma gaat fout. Op welk moment gebeürt dat? Ook in dit voorbeeld moeten we ervoor zorgen dat X niet buiten zijn boekje gaat. Dat betekent: X mag niet groter worden dan 4. Voeg de volgende regel aan het programma toe:

```
85 IF X > 4 THEN STOP
```

Het probleem is daarmee opgelost. Het kan ook zo:

```
60 FOR X=1 TO 4
70 PRINT X;A(X);A$(X)
80 NEXT
90 STOP
```

Met FOR en NEXT kunnen we een hele tabel afwerken. We kunnen natuurlijk weer gebruikmaken van alle mogelijkheden die FOR en NEXT bieden, zoals:

```
60 FOR X=1 TO 4 STEP 2
60 FOR X=4 TO 1 STEP -1
```

Haal de regels 60 t/m 90 met een DELETE-opdracht van het programma af en vervang ze door de volgende programmaregels:

```
60 INPUT "LETTER";L$
70 FOR X=1 TO 4
80 IF A$(X)=L$ THEN 120
90 NEXT
100 PRINT L$ " NIET GEVONDEN"
110 GOTO 60
120 PRINT L$ " STAAT IN ELEMENT" X
130 PRINT "DAARBIJ HOORT IN TABEL A:" A(X)
140 GOTO 60
```

Op regel 60 wordt een letter ingevoerd in L$. Vervolgens gaan we **zoeken** in de tabel A$ of deze letter voorkomt. Als dat niet het geval is loopt de MSX de hele tabel langs, vindt niets, komt op regel 100 en zet op het scherm dat de letter niet gevonden werd. Daarna gaan we terug naar regel 60. Vindt de MSX de letter wel op regel 80, dan springt hij naar regel 120 en zet op het scherm in welk element de letter staat. Regel 130 laat zien dat we ook een verbinding kunnen leggen tussen twee tabellen. We komen daarop later terug.

Regel 60 kan nog mooier:

```
60 PRINT "LETTER: ";:INPUT$(1)
```

We hebben dan geen RETURN nodig bij het typen van de gezochte letter.

102. Ogen tellen in tabellen

Hoe kunnen we zo'n programma als "ogen tellen" eenvoudiger maken met behulp van tabellen? In dit programma kan dat heel eenvoudig door gebruik te maken van een toevallige omstandigheid. Die omstandigheid is dat we de MSX ogen laten gooien van 1 t/m 6. Die kunnen we dan mooi tellen in tabel-elementen van 1 t/m 6 van een tabel. Gooit de MSX 4, dan tellen we dat in tabel-element 4. We hoeven dan niet meer te vragen hoeveel ogen de MSX gooide om uit te maken in welk vakje dat geteld moet worden, maar we tellen het eenvoudig in het tabel-element dat hoort bij het aantal ogen dat de MSX gegooid heeft. De centrale opdracht in dit programma is dus (er staat nog geen regelnummer voor want we weten nog niet precies waar deze regel terechtkomt):

T(OGEN)=T(OGEN)+1 (voor de duidelijkheid gebruiken we een naam van meer letters)

We tellen in T(OGEN) het aantal keren dat de MSX OGEN gooit, dus:

tel	in	T(1)	het	aantal	keren	dat	de	MSX 1	gooit
"	"	T(2)	"	"	"	"	"	" 2	"
"	"	T(3)	"	"	"	"	"	" 3	"
"	"	T(4)	"	"	"	"	"	" 4	"
"	"	T(5)	"	"	"	"	"	" 5	"
"	"	T(6)	"	"	"	"	"	" 6	"

We gaan nu het programma opbouwen. De eerste regel is nieuw:

10 DIM T(6)

Deze DIM-opdracht zorgt ervoor dat we een tabel krijgen van 7 elementen, genummerd van 0 t/m 6. Alleen de elementen van 1 t/m 6 gaan we gebruiken. T(0) doet voor spek en bonen mee. We maken nu het scherm schoon en laten de dobbelsteen rollen:

20 CLS
30 FOR I=1 TO 1000
40 OGEN=INT(RND(5)*6)+1

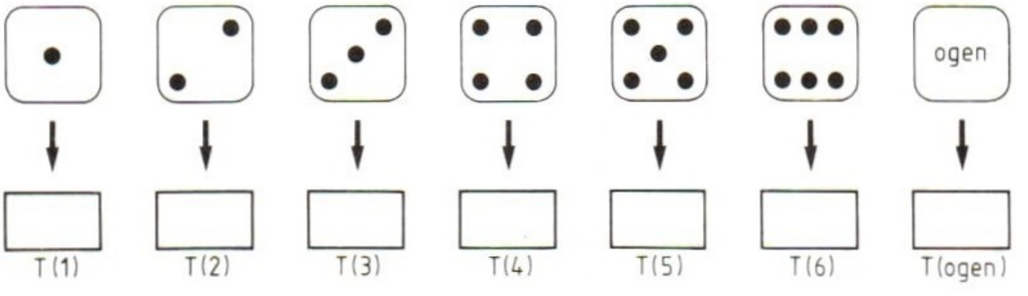
Op regel 50 kunnen we nu tellen:

50 T(OGEN)=T(OGEN)+1 (haakjes op de juiste plaats!)
60 NEXT

De MSX kan niet buiten zijn indexboekje gaan: in OGEN komt altijd een getal van 1 t/m 6. Nu moeten de resultaten nog op het scherm komen:

```
70 PRINT TAB(10) "OGEN" TAB(20) "AANTAL MALEN"
80 FOR OGEN=1 TO 6
90 PRINT TAB(11) OGEN TAB(24) T(OGEN)
100 NEXT
110 STOP
```

Op regel 70 zetten we een kopregel op het scherm. Met de TAB bepalen we de plaats. Daarna gaan we zes elementen van de tabel T op het scherm zetten. Dat doen we op het rijtje af. OGEN gebruiken we als index. Die laten we lopen van 1 t/m 6. Zo werken we alle zes tabel-elementen af tot het overzicht op het scherm staat. Neem het programma over in het geheugen van de MSX en voer het uit. Zet het, als het goed werkt, op cassette.



103. Tabelproblemen

In deze paragraaf geven we voorbeelden van kleine probleempjes met tabellen. Deze voorbeelden zijn geen volledige programma's. Als we ze desondanks uitvoeren zien we niet precies wat er gebeurt.
Eerst gaan we alle elementen van een tabel 0 maken. Als we beginnen met de uitvoering van een programma staat in alle tabel-elementen vanzelf 0, maar soms is het nodig dat in het programma ook nog eens te doen. We gaan uit van een tabel van 100 elementen die we zo op nul zetten:

```
DIM A(100), A$(100)
FOR I=1 TO 100:A(I)=0:NEXT
```

Op dezelfde manier kunnen we in een tabel van strings alle elementen een bepaalde waarde geven:

```
FOR I=1 TO 100:A$(I)="MSX":NEXT
```

Het optellen van 100 getallen in de tabel A kan bijvoorbeeld zo:

```
S=0:FOR I=1 TO 100:S=S+A(I):NEXT
```

Als we in een programma een tabel met getallen nodig hebben kunnen we die tabel in het programma zo vullen:

```
A(1)=40:A(2)=30:A(3)=70:A(4)=45:A(5)=23 enz.......A(100)=87
```

Vaak is het gemakkelijker om dat met behulp van een READ-opdracht en dataregels te doen. Dus op deze manier (we kunnen dit natuurlijk alleen uitproberen als we ook werkelijk 100 getallen op dataregels zetten!):

```
FOR I=1 TO 100:READ A(I):NEXT
DATA 40,30,70,45,23,enz..........,87
```

Datzelfde kan natuurlijk ook voor tabellen met strings:

```
FOR I=1 TO 100:READ A$(I):NEXT
DATA JAN,PIET, enz...................,WIM
```

Op de volgende manier kunnen we twee tabellen, één voor getallen en één voor teksten vullen via het toetsenbord:

FOR I=1 TO 100:INPUT "nummer";A(I):INPUT "naam";A$(I):NEXT of mooier:

```
FOR I=1 TO 100:PRINT "nummer"I;:INPUT A(I):PRINT "naam"I;:INPUT A$(I):NEXT
```

Bij dit voorbeeld moeten we wel een opmerking maken. De getallen en teksten die hier getypt worden en opgeborgen worden in tabel-elementen staan alleen tijdens de uitvoering van het programma in het geheugen. Als we het programma opnieuw uitvoeren zijn de tabellen weer leeg. Vullen via het toetsenbord met INPUT-opdrachten is dus iets anders dan de hele inhoud van de tabel in het programma opnemen zoals bijvoorbeeld met een READ-opdracht kan.

Soms moeten we uit een rij getallen het grootste of het kleinste getal zoeken. Voor het kleinste getal gaat dat zo:

```
K=A(1)
FOR I=2 TO 100
IF A(I)< K THEN K=A(I)
NEXT
```

We beginnen hier gewoon bij het eerste getal in de rij. Dat beschouwen we als het kleinste dat we tot nu toe gevonden hebben en zetten het in de variabele K. Daarna gaan we de rest van de tabel langs. We beginnen bij nummer 2 en kijken of het getal in element 2 misschien kleiner is dan het getal dat we tot nu toe als het kleinste beschouwd hebben en dat in K staat. Als dat niet zo is gaan we gewoon door tot we alle getallen gehad hebben. Vinden we een getal dat kleiner is dan het kleinste getal tot nu toe, dan vervangen we het getal in K door het nog kleinere getal dat we gevonden hebben. Zo wordt de hele rij afgewerkt.

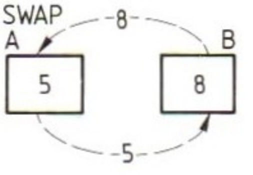
Het zoeken van het grootste getal in de rij is natuurlijk bijna hetzelfde. In het volgende voorbeeld willen we twee dingen weten: wat is het grootste getal en op welke plaats staat dat grootste getal in de tabel. Het grootste getal zelf komt in GA, de index die daarbij hoort in GI:

```
GA=A(1):GI=1
FOR I=2 TO 100
IF A(I)> GA THEN GA=A(I):GI=I
NEXT
```

104. Een tabel sorteren

In veel gevallen is het handig een tabel te kunnen sorteren. Daarvoor kunnen we de volgende subroutine gebruiken, die natuurlijk ook op andere regels mag staan:

```
9000 FOR I=B+1 TO L
9010 FOR J=B TO I-1
9020 IF A(J) < A(I) THEN NEXT J:GOTO 9040
9030 SWAP A(J),A(I):NEXT J
9040 NEXT I
9050 RETURN
```



Een korte toelichting:

a. I en J zijn willekeurige namen voor de index. B is het eerste element dat gesorteerd moet worden, L het laatste.
b. De routine gebruikt tabel A, maar is natuurlijk ook voor elke andere tabel (ook voor stringtabellen) geschikt. De naam moet dan aangepast worden.
c. Als we twee tabellen willen sorteren op basis van één van beide (b.v. een tabel A$ met namen, sorteren op nummer in tabel A) moet de bijbehorende tabel opgenomen worden in 9030:

```
9030 SWAP A(J),A(I):SWAP A$(J),A$(I):NEXT J
```

De opdracht SWAP zorgt ervoor, dat de inhoud van de <u>twee</u> genoemde variabelen verwisseld wordt.

We kunnen de routine testen met het volgende programma, waarin we de MSX zelf een reeks te sorteren getallen laten "genereren":

```
10 CLEAR 500
20 INPUT "HOEVEEL GETALLEN";N
30 DIM A(N)
40 FOR I=1 TO N:A(I)=INT(RND(5)*1000):PRINT A(I);:NEXT I
50 PRINT:PRINT:B=1:L=N:GOSUB 9000
60 FOR I=1 TO N:PRINT A(I);:NEXT I
70 PRINT:PRINT
80 ERASE A
90 GOTO 20
```

De opdracht ERASE A op regel 80 "wist" de tabel A uit. Daardoor kunnen we op regel 30 de tabel opnieuw "DIMmen". Zonder ERASE zouden we dan het bericht Redimensioned array in 30 krijgen.

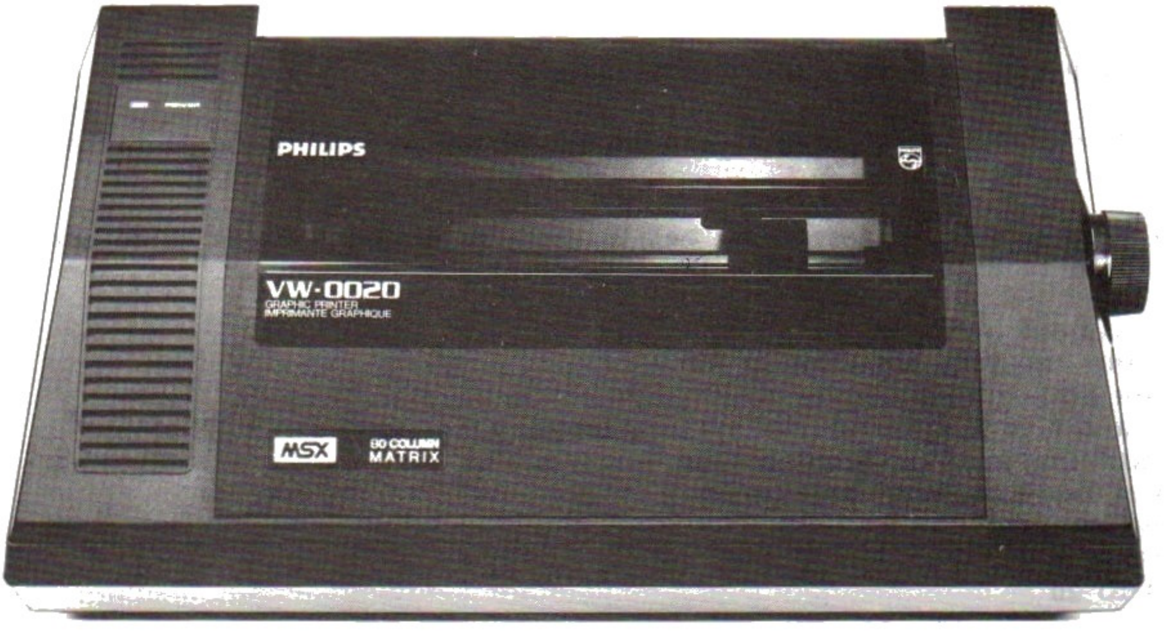
Om een idee te hebben van de tijdsduur: 20 getallen sorteert de MSX in bijna 2 sec., over 100 getallen doet hij ± 45 seconden, voor 1000 getallen heeft hij met <u>deze</u> sorteerroutine een half uur nodig.

Opmerking: de MSX heeft een ingebouwde klok, die TIME heet. We kunnen deze klok op 0 zetten (TIME=0) en we kunnen op de klok kijken "hoe laat het is" (PRINT TIME of bijvoorbeeld A=TIME).
TIME is geen "nette" klok. De MSX telt in deze variabele met stapjes van 1/50 seconde tot hij bij 65535 is. Dan begint hij opnieuw met 0.
We kunnen de tijd in seconden in ons sorteerprogramma meten, bijvoorbeeld zo:

```
9000 TIME=0:FOR I=B+1 TO L
```

9050 PRINT "Tijd was" TIME/50 "seconde":RETURN

Bij het begin van het sorteren zetten we de klok op 0, als de MSX klaar is zetten we de stand van TIME gedeeld door 50 op het scherm. Dat geeft de tijd in seconden, behalve als de MSX langer dan 21 minuten bezig is geweest.
In dat geval is hij alweer met 0 begonnen.



105. Tabellen in programma's

Tabellen kunnen op veel manieren in programma's gebruikt worden. Het is onmogelijk al die manieren in dit boek te laten zien. We beperken ons tot twee voorbeelden.
Eerst laten we zien hoe twee kleine tabellen met elkaar vergeleken worden.

In het eerste voorbeeld gaan we twee tabellen van zes getallen met elkaar vergelijken. Dat zijn de tabellen A en B. In tabel A staan zes getallen tussen 1 en 41. We laten de MSX die getallen willekeurig uitzoeken. In tabel B komen ook zes getallen tussen 1 en 41 te staan. Deze getallen typen we bij een INPUT-opdracht. We laten daarna de MSX uitzoeken hoeveel getallen uit de tabel A ook in de tabel B staan. We kunnen het ook zo zeggen: we laten de MSX zes getallen uitzoeken tussen 1 en 41. Die zet hij in tabel A. Daarna gaan wij raden welke zes getallen de MSX opgezocht heeft. De getallen die wij raden zetten we in tabel B. Daarna zoekt de MSX uit hoeveel getallen wij goed geraden hebben. Een voorbeeld:

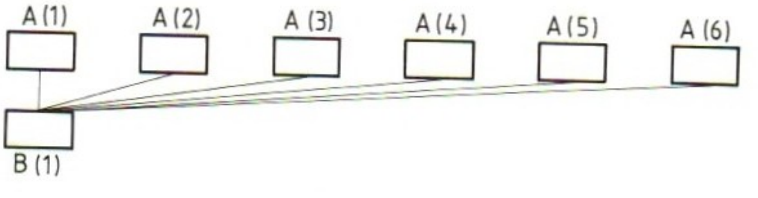
MSX	:	A(1)	A(2)	A(3)	A(4)	A(5)	A(6)
		22	8	41	7	1	16
Wij	:	B(1)	B(2)	B(3)	B(4)	B(5)	B(6)
		7	33	11	19	8	24

Dus : 2 goed, want B(1)=A(4) en B(5)=A(2)

Om te kunnen bepalen hoeveel getallen goed geraden zijn moeten we <u>alle</u> getallen uit de tabel B vergelijken met <u>alle</u> getallen uit de tabel A. Dat werkt zo:

P loopt van 1 tot 6, I=1, vergelijk A(P) met B(I), dus:

```
Vergelijk A(1) met B(1)
    "     A(2)  "  B(1)
    "     A(3)  "  B(1)
    "     A(4)  "  B(1)
    "     A(5)  "  B(1)
    "     A(6)  "  B(1)
```



P loopt van 1 tot 6, I=2, vergelijk A(P) met B(I), dus:

```
Vergelijk A(1) met B(2)
    "     A(2)  "  B(2)
    "     enz.
    "     A(6)  "  B(2) enzovoorts!
```

In het programma gebruiken we daarvoor de volgende opdrachten:

```
180 T=0
190 FOR I=1 TO 6
200 FOR P=1 TO 6
210 IF A(P)=B(I) THEN T=T+1
220 NEXT P
230 NEXT I
240 PRINT T "goed"
```

T gebruiken we om te tellen hoeveel getallen goed geraden zijn. Die zetten we eerst op nul. Elke keer als we op regel 200 twee gelijke getallen vinden tellen we 1 bij T op.

Opmerking: De regelnummers passen precies in het programma dat we straks maken.
De MSX moet zes getallen uitzoeken voor de tabel A. Dat laten we hem zo doen:

```
30 FOR I=1 TO 6
40 A(I)=INT(RND(5)*40)+1
50
60 NEXT
```

In dit stukje programma is één regel opengelaten. Daar komen we direct op terug. De getallen in de tabel B moeten we zelf typen. Dat doen we in dit programma zo:

```
70 CLS
80 PRINT "Geef 6 hele getallen van 1 t/m 41"
90 FOR I=1 TO 6
100 PRINT "GETAL"I"=";
110 INPUT B(I)
120 IF B(I) < 1 OR B(I) > 41 THEN PRINT "fout, ander getal":GOTO 100
130
140
150
160
170 NEXT I
```

In dit stukje zorgen we er in de eerste plaats voor dat de getallen in de tabel B tussen 1 en 41 (inclusief de grenzen) liggen. In de vier open regels moeten we nog een probleem oplossen. In de tabel B moeten zes verschillende getallen staan. Een getal mag dus niet dubbel in deze tabel voorkomen. We kunnen daarvoor zorgen door het getal dat we typen te vergelijken met alle getallen in de tabel B die we al getypt hebben. Dat gaat alleen voor het eerste getal niet op. Dat kunnen we nergens mee vergelijken. Dat is niet erg, want het kan ook nog niet dubbel in de tabel voorkomen.

Dit kunnen we zo aanpakken:

```
130 IF I=1 THEN 170
140 FOR P=1 TO I-1
150 IF B(P)=B(I) THEN PRINT "Dubbel; ander getal":GOTO 100
160 NEXT P
```

In de tabel A mogen natuurlijk ook geen dubbele getallen zitten. Daarvoor hadden we regel 50 al opengelaten. We zullen dit probleem op een andere manier oplossen.
We gebruiken nog een derde tabel, de tabel C. In die tabel laten we de MSX de getallen van 1 t/m 41 zetten. Als de MSX op regel 40 een getal uitzoekt laten we hem in de tabel C aantekenen dat hij dit getal gehad heeft. Dat doen we door op de juiste plaats in tabel C het getal nul te zetten. Overal in tabel C waar een nul staat is dus vastgelegd dat de MSX dit getal al gehad heeft.
De volgende opdrachten zijn daarvoor nodig:

```
20 FOR J=1 TO 41:C(J)=J:NEXT
50 IF C(A(I))=0 THEN 40 ELSE C(A(I))=0
```

De bedoeling van dit stukje kunnen we zo weergeven:

Als A(3) bijvoorbeeld 21 wordt controleren we:
is C(21) al 0? dan hebben we 21 al gehad; terug naar regel 40
is C(21) nog geen 0? dan is 21 goed en maken we C(21) = 0, zodat 21 niet nog eens kan voorkomen; we gaan verder op regel 60.

Op regel 50 staat iets bijzonders. We gebruiken hier een <u>element uit de tabel A als index</u> bij de tabel C. We kunnen dat zo lezen:

C(A(I)) is tabel C, element A(I); A(I) is tabel A, element I. We mogen het ook zo zeggen: C(A(I)) is tabel C, elementnummer staat in tabel A, element I.

Met een DIM-opdracht kunnen we het programma nu volledig maken. Zet het in de MSX, probeer of het goed werkt en leg het vast op een cassette.

Opmerkingen:

a. Op regel 210 kunnen we gebruikmaken van de wetenschap dat elk getal in elk van de twee tabellen maar één keer voor kan komen. Als we dus gevonden hebben dat een getal in tabel A gelijk is aan een getal in tabel B, hoeven we voor dat getal niet verder te zoeken. Regel 210 mag er dus ook zo uitzien:

   ```
   210 IF A(P)=B(I) THEN T=T+1:GOTO 230
   ```

b. Als we een nieuw programma moeten uitproberen is het soms erg handig een paar PRINT-opdrachten in het programma te zetten, die we er later weer uithalen. In dit programma kunnen we bijvoorbeeld niet zien welke getallen de MSX in de rij A heeft gezet. We kunnen dus ook niet controleren of hij de vergelijkingen goed maakt. We kunnen dat oplossen door voorlopig de volgende opdrachten in het programma op te nemen:

```
235 FOR I=1 TO 6:PRINT A(I);:NEXT:PRINT
236 FOR I=1 TO 6:PRINT B(I);:NEXT:PRINT
```

Deze opdrachten zorgen ervoor dat we de twee tabellen A en B te zien krijgen, zodat we zelf kunnen nagaan hoeveel getallen goed geraden zijn. Als het programma goed werkt halen we deze opdrachten gewoon weer weg.

c. Twee NEXT-opdrachten na elkaar mogen ook zo gegeven worden:

```
160 NEXT P,I          (170 weghalen)
220 NEXT P,I          (230 weghalen)
```

Ook goed, maar minder duidelijk (voor ons, niet voor de MSX) is:

```
160 NEXT:NEXT
220 NEXT:NEXT
```

Hier komt het volledige programma:

```
10 DIM A(6),B(6),C(41)
20 FOR I=1 TO 41:C(I)=I:NEXT
30 FOR I=1 TO 6
40 A(I)=INT(RND(5)*40)+1
50 IF C(A(I))=0 THEN 40 ELSE C(A(I))=0
60 NEXT I
70 CLS
80 PRINT "Geef 6 getallen tot 41"
90 FOR I=1 TO 6
100 PRINT "GETAL"I"=";
110 INPUT B(I)
120 IF B(I) < 1 OR B(I) > 41 THEN PRINT "fout; ander getal":GOTO 100
130 IF I=1 THEN 170
140 FOR P=1 TO I-1
150 IF B(P)=B(I) THEN PRINT "dubbel; ander getal":GOTO 100
160 NEXT P
170 NEXT I
180 T=0
190 FOR I=1 TO 6
200 FOR P=1 TO 6
210 IF A(P)=B(I) THEN T=T+1
220 NEXT P
230 NEXT I
240 PRINT T "goed"
250 PRINT:GOTO 80
```

106. Opdrachten

Opdracht 2:
In de tabel A(1) tot en met A(100) staan getallen. Met welke regel(s) drukken we elk tiende getal in de rij af?

```
a. FOR I=1  TO 10:PRINT A(I*10):NEXT
b. FOR I=1  TO 10:PRINT A(I)*10:NEXT
c. FOR I=10 TO 100 STEP 10:PRINT A(I):NEXT
d. FOR I=10 TO 100 STEP 10:PRINT I:NEXT
```

Opdracht 3:
Zet met de opdracht

FOR I=1 TO 60:A(I)=100+I:NEXT

getallen in de tabel A(1) t/m A(60). Zet daarna de getallen op het scherm, maar zo, dat steeds 3 getallen op één regel komen.
Voorbeeld: A(1)=101, A(2)=102 enz.

```
          101    102    103
          104    105    106
          107    enz.
```

Opdracht 4:
Zet met de opdracht

FOR I=1 TO 60:A(I)=100+I:NEXT

getallen in de tabel A(1) t/m A(60). Zet daarna getallen op het scherm, maar zo, dat de eerste 20 getallen onder elkaar komen, de volgende 20 daarnaast en zo verder.
Voorbeeld: A(1)=101, A(2)=102 enz.

```
101    121    141
102    122    142
103     '     143
 '      '      '
 '      '      '
120    140    160
```

Opdracht 5:
Letters tellen. Maak een programma, waarin de verschillende letters in een getypte tekst worden geteld. Gebruik voor het tellen een tabel T(1) t/m T(26). In T(1) worden de letters A en a geteld, in T(2) de letters B en b, in T(3) C en c enzovoorts. Lees- en andere tekens worden niet geteld.
De tekst mag uit meer regels bestaan. Het telresultaat wordt zichtbaar als we een teken # typen.

Aanwijzing: Ga uit van de ASCII-code van de letters. A is code 65, Z is code 90, a is code 97, z is code 122. Zoek dus een verband tussen:

letter	code	tellen in
A	65	T(1)
B	66	T(2)
C	67	T(3)
enz. t/m		
Z	90	T(26)

én

letter	code	tellen in
a	97	T(1)
b	98	T(2)
enz. t/m		
z	122	T(26)

HOOFDSTUK 13: Een programma met een bestand

107. Bestanden

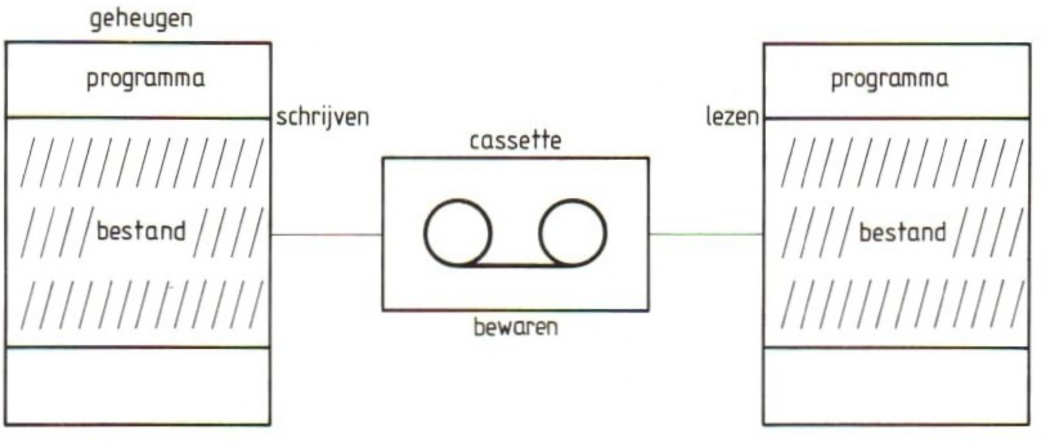
De MSX kan méér met de cassette dan alleen programma's opslaan (CSAVE) en laden (CLOAD en CLOAD?). De cassette heeft de functie van een "extern geheugen": een stuk geheugen buiten de computer, om dingen te bewaren. Dat heeft twee voordelen:

a. wat op cassette staat gaat niet verloren als de computer uitgeschakeld wordt (wat in het interne MSX-geheugen staat wél);
b. we kunnen veel meer bewaren dan in het interne geheugen van de MSX kan staan.

Dat geldt voor programma's, dat geldt ook voor andere informatie. We kunnen op cassette ook getallen en teksten opslaan. In dat geval spreken we van bestanden.

Het gebruik van de cassette voor het opslaan en gebruiken van bestanden heeft wel zijn beperkingen:

1. We kunnen een bestand in zijn geheel óf lezen, óf schrijven. Anders gezegd:



   We kunnen een hele reeks gegevens (getallen en/of teksten) uit het geheugen naar de cassette brengen (een bestand schrijven) of een hele reeks gegevens (getallen en/of teksten) van de cassette naar het interne geheugen brengen (één bestand lezen).
2. Wat dus niet kan is bijvoorbeeld: een woord typen, de MSX dat woord op cassette laten zetten, dan weer een woord typen enz. We moeten, als we getypte woorden op cassette willen zetten eerst alles typen, in het geheugen bewaren en dan de hele reeks als één bestand op cassette schrijven.
3. Wat ook niet kan is afwisselend lezen en schrijven, bijvoorbeeld: lees een getal van cassette, tel er 10 bij op en schrijf de uitkomst weer op cassette. Ook hier werkt het alleen zo: lees eerst het hele bestand getallen, pas daarna de getallen aan en schrijf daarna het gehele bestand weer op de cassette weg. ("Wegschrijven" is het gebruikelijke woord).

We kunnen niets proberen als we niet eerst een bestand maken. We zullen moeten beginnen met het maken van een bestand in het geheugen dat we vervolgens op een cassette wegschrijven. We maken het ons niet te moeilijk en gaan uit van een tabel:

```
10 CLEAR 500
20 DIM A(100)
30 FOR I=1 TO 100
40 A(I)=I
50 NEXT I
60 PRINT "Het bestand bevat de volgende getallen:"
70 FOR I=1 TO 100
80 PRINT A(I)
90 NEXT I
```

Voer het programma uit. We zien dan dat de zin "Het bestand bevat... enz." niet mooi op het scherm komt: de laatste n staat op een nieuwe regel. We hebben hiervoor drie oplossingen:

a. We verdelen de zin in twee stukken, zodat de tekst wel goed op het scherm komt, bijvoorbeeld:

   ```
   60 PRINT "Het bestand bevat de volgende getal-":PRINT "len:"
   ```

   De twee PRINT-opdrachten zorgen voor een nette tekst.

b. We zetten in de tekst zoveel spaties als nodig is om de tekst behoorlijk over twee regels te verdelen:

   ```
   60 PRINT "Het bestand bevat de volgende getal- len:"
   ```

   De spatie tussen het streepje en de l komt precies op de laatste positie van het scherm.

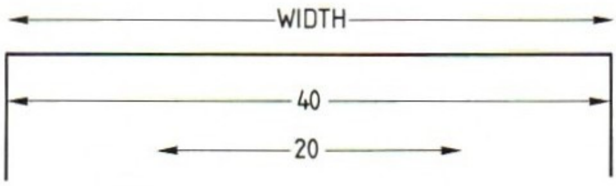
c. We maken het scherm breder!

We hebben gezien dat op het scherm 37 tekens op één regel passen, maar dat aantal kunnen we veranderen met de opdracht WIDTH (breedte). Het aantal tekens dat op een regel past is afhankelijk van de schermsoort. Scherm 0 (zonder randen) kan maximaal 40 tekens breed gemaakt worden. Scherm 1 (met randen) kan hooguit 32 tekens breed zijn. We regelen dat met de opdracht WIDTH. Probeer:

```
55 SCREEN 0:WIDTH 40
```

Merk op dat de SCREEN-opdracht al een CLS inhoudt. Probeer daarna enkele andere mogelijkheden:

```
55 SCREEN 0:WIDTH 20
55 SCREEN 1
55 SCREEN 1:WIDTH 32
```



Merk op dat de breedte bij de schermsoort hoort. Als we SCREEN 1 breder maken, blijft SCREEN 0 zijn oude breedte behouden.

Probeer ook de regels te overtreden:

```
55 SCREEN 0:WIDTH 50
55 SCREEN 1:WIDTH 0
```

Zet scherm 0 op een breedte van 40 tekens (in het programma); dan keren we terug naar ons bestand.

Het programma kan nu een bestand maken. We willen dit bestand op cassette schrijven. Daarvoor moeten we drie dingen doen:

a. we moeten de MSX meedelen dat we een bestand willen gebruiken en waarvoor. Daarvoor dient de opdracht OPEN;
b. we moeten in het geopende bestand 100 getallen schrijven. Daarvoor gebruiken we de opdracht PRINT#. Een variant van de gewone PRINT-opdracht dus;
c. we moeten de MSX meedelen dat we klaar zijn met het schrijven van het bestand. Daarvoor gebruiken we de opdracht CLOSE.

Vul het programma zo aan en lees daarna de toelichting:

```
200 OPEN "CAS:tabel" FOR OUTPUT AS #1
210 FOR I=1 TO 100
220 PRINT #1,A(I);
230 NEXT I
240 CLOSE #1
```

Let op de afwijkende regelnummers: we zorgen zo voor wat ruimte, die we direct nog nodig hebben.

Op regel 200 staat de OPEN-opdracht, waarin we de MSX vertellen dat we een bestand gaan gebruiken. Dat bestand heeft twee kenmerken:

1. In het programma heeft het een nummer 1. Dat nummer dient om de MSX te laten weten welke opdrachten voor dit bestand bestemd zijn. Nummer 1 komt terug in de opdrachten PRINT#1 en CLOSE#1. Het nummer is dus alleen in dit programma van belang. Hetzelfde bestand mag in een ander programma best een ander nummer krijgen.

2. Op de cassette heeft het bestand een naam: tabel. Deze naam dient om de MSX te laten weten hoe het bestand op cassette tussen de andere bestanden en programma's terug te vinden is. De regels voor deze naam zijn bekend: begin met een letter, maximaal 6 tekens.

In de OPEN-opdracht staan verder nog twee mededelingen:

3. dat het over een bestand op cassette gaat: CAS;
4. dat we het bestand willen gaan schrijven: OUTPUT
   Het bestand gaat vanuit het geheugen naar de cassette en wordt dus uitgevoerd.

De andere mogelijkheden van de OPEN-opdracht zullen we nu niet bespreken.

Op regel 210-230 worden de honderd getallen van de tabel A op de cassette gezet met een PRINT-opdracht. Aan het hekje (#) ziet de MSX dat het om een schrijfopdracht voor een bestand gaat en niet om een gewone PRINT-opdracht. Aan het cijfer 1 ziet de MSX dat deze opdracht bij het bestand "tabel" op cassette hoort.

Op regel 240 tenslotte delen we de MSX mee, dat we klaar zijn met schrijven: alle 100 getallen zijn opgenomen op de band en het bestand kan weer gesloten worden.

We kunnen het programma nu uitproberen, maar... we moeten wel even goed aan de bediening van de cassetterecorder denken. We moeten tijdig de recorder op "opnemen" zetten. Daarom hebben we in de regelnummers wat ruimte gelaten:

```
100 PRINT
110 PRINT "Recorder op opnemen s.v.p."
120 PRINT "(toets)";:INPUT$(1)
```

Op opdracht PRINT is nodig om ervoor te zorgen dat de tekst van regel 110 niet achter het laatste getal uit de tabel, maar netjes op een nieuwe regel komt. De INPUT$-opdracht dient alleen om de MSX even stil te zetten: hij wacht tot er een toets wordt aangeslagen, voordat hij met de volgende opdracht (het openen van het bestand) verder gaat. We moeten op het scherm natuurlijk wel kunnen zien dat de computer op een toets staat te wachten. Vandaar de tekst "(toets)" op regel 120.

Na het opnemen (schrijven) van het bestand mogen we vooral niet vergeten de opnameknop van de recorder weer uit te zetten, anders lopen we de kans het hele bestand even snel weer kwijt te raken. Daarom:

```
250 PRINT
260 PRINT "tabel geschreven; recorder uit"
270 PRINT "(toets)";:INPUT$(1)
280 STOP
```

Opmerking: deze maatregelen in het programma zijn ook zinvol als de MSX de cassetterecorder zelf op afstand bedient. De computer zorgt niet zelf voor het indrukken en weer uitschakelen van de opneemtoets en dat is o zo gemakkelijk vergeten.

108. Een bestand lezen

We willen nu natuurlijk ook graag weten of het bestand werkelijk op de cassette staat. Daarvoor moeten we het bestand lezen. Zet eerst het schrijfprogramma op een cassette, zodat we het later nog eens kunnen proberen. Maak daarna een nieuw programma (dat kan gedeeltelijk uit het oude "ge-edit" worden):

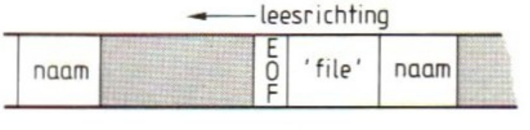
```
10 CLEAR 500
20 DIM B(100)
30 SCREEN 0:WIDTH 40
40 PRINT "Recorder op weergeven s.v.p."
50 PRINT "(toets)";:INPUT$(1)
60 OPEN "CAS:tabel" FOR INPUT AS #1
70
80 INPUT #1,B(I)
90 GOTO 70
100 CLOSE #1
110 PRINT "Recorder stoppen s.v.p."
120 PRINT "(toets)";:INPUT$(1)
130 PRINT "Gelezen getallen:"
140 FOR I=1 TO 100
150 PRINT B(I);
160 NEXT
170 STOP
```

Regel 70 is nog even open gelaten om een probleem op te kunnen lossen. Verder is dit programma het "omgekeerde" van het vorige. Toelichting is dan ook niet nodig.

Het probleem dat we nog moeten oplossen, lijkt op het "Out of data"-probleem, dat we eerder tegenkwamen. De MSX krijgt steeds de opdracht een getal te lezen uit het bestand. Elke keer wordt hij door de GOTO op regel 90 teruggestuurd om het volgende getal op te halen. Dat lukt 99 keer, maar bij de honderdste keer gaat het mis: de MSX moet het 101-ste getal lezen en dat zit niet meer in het bestand. De MSX meldt dat met het bericht "Input past end" (ik moet iets lezen, maar ik ben het eind van het bestand al voorbij).

Nu is de oplossing van dit probleem niet moeilijk: de computer merkt dat hij bij het eind van het bestand gekomen is en het enige wat hij van ons verlangt, is dat wij hem vertellen wat hij in dat geval moet doen:

```
70 IF EOF(1) THEN 100
```



EOF is de afkorting van "end of file" (file = bestand). Op deze regel staat dus: als je het einde van bestand 1 (het nummer tussen haakjes) tegenkomt, ga je verder met regel 100. De regel is simpel: **zet vóór elke lees-opdracht (INPUT) een IF EOF-opdracht** zodat de computer weet wat hij bij het bereiken van het einde van het bestand moet doen.

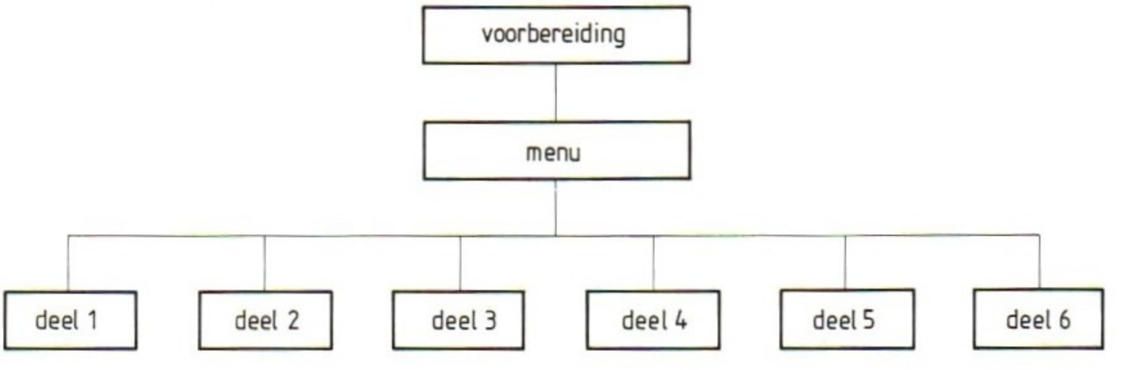
Opmerkingen:

1. Er zijn nog veel meer mogelijkheden om met bestanden te werken. De MSX kan meer bestanden in het programma verwerken; in bestanden kunnen ook teksten staan. Het voert echter te ver al deze mogelijkheden in dit boek te bespreken.
2. Het werken met bestanden vraagt nogal wat handelingen met de cassetterecorder. Daarbij kan zoveel mis gaan door bijvoorbeeld het niet tijdig (te laat of te vroeg) stoppen en starten van de recorder, de opneem-toets vergeten e.d., dat het onmogelijk is in dit boek alles te bespreken. Als u met bestanden wilt werken is het advies: lees de documentatie bij de computer zeer zorgvuldig door en probeer stapje voor stapje. Dat kost (door het langzame werken van de cassette) veel tijd, maar is toch nodig.
3. Essentiëel bij het werken met bestanden is, dat twee activiteiten elkaar beïnvloeden. Als we bij het lezen van een bestand merken dat het programma niet oplevert wat wij ervan verwachten, kan dat aan het leesprogramma liggen. Maar het kan ook zijn dat er met de cassetterecorder of de cassette zelf iets mis is. Het is echter ook heel goed mogelijk dat we een fout gemaakt hebben in het programma dat het bestand geschreven heeft. Juist dát maakt het programmeren van programma's, die bestanden op cassette gebruiken, zo lastig en vooral tijdrovend.

109. Hoofdprogramma

Het programma dat we hier bespreken kunnen we gebruiken om de voorraad bij te houden. Op dataregels zetten we artikelnummers en artikelomschrijvingen. In het programma brengen we deze gegevens over naar een tabel. De informatie over de voorraad zetten we eveneens in een tabel. Deze tabel schrijven we op een cassette. We kunnen elke keer de tabel van de cassette overbrengen naar het geheugen. Het programma zorgt ervoor dat we de voorraad kunnen zien en aanpassen.
Daarna wordt de veranderde tabel weer op de cassette gezet. De structuur van dit programma is zo opgebouwd:

a. Het programma begint met een aantal voorbereidende opdrachten. Hier horen bijvoorbeeld de opdrachten CLEAR en DIM thuis.
b. Vervolgens komt er een keuzelijst, een zogenaamd "menu". Dat is een lijst met mogelijkheden van het programma op het scherm. Dit is het centrale punt in het programma.
   Na iedere actie komt het programma hier terug en kunnen we een nieuwe keuze maken.
c. Voor elk van de mogelijke activiteiten is er een stuk programma. Na de keuze laten we de MSX naar het gekozen programmadeel toe springen, voeren dat uit en gaan daarna terug naar de centrale keuze. We kunnen deze structuur weergeven in het volgende schema:



Elk programma begint met een voorbereidend gedeelte. Daarin zit in ieder geval een aantal vaste activiteiten, dat in elk programma terugkomt;

a. een CLEAR-opdracht voor het reserveren van werkruimte voor strings;
b. een SCREEN-opdracht en een WIDTH-opdracht om voor het juiste scherm te zorgen, eventueel met een COLOR-opdracht en een KEY OFF om de laatste regel vrij te krijgen.

Verder gebruiken we de voorbereidingen voor het "dimensioneren" van tabellen en voor het toekennen van waarden aan variabelen.

Maak dit stukje programma:

```
10 REM **VOORRAAD**
20 CLEAR 1000
30 SCREEN 0:WIDTH 40:COLOR 1,14:KEY OFF
40 AA=50:DIM A$(AA),A(AA),V(AA)
50 S$="N":REM S$ geeft de status van het programma: N=tabel niet gelezen,
L=tabel gelezen, G=tabel geschreven
60 KOP$="nummer      naam                    voorraad"
```

Op regel 40 worden drie tabellen opgegeven. Het aantal elementen van de tabellen is aangegeven in de vorm van een variabele die aan het begin van regel 40 de waarde 50 heeft gekregen. Deze methode biedt voordelen als we nog niet zeker zijn van het aantal elementen dat we in een tabel wensen. De variabele AA komt niet alleen voor in de DIM-opdracht maar ook op een aantal andere plaatsen in het programma, bijvoorbeeld daar waar de tabel met omschrijvingen gelezen wordt. Wanneer we nu het aantal elementen willen veranderen en bijvoorbeeld het programma geschikt willen maken voor honderd artikelen, hoeven we alleen op regel 40 de waarde van de variabele AA te wijzigen. De rest van het programma is dan "vanzelf" aangepast.

We gaan de drie opgegeven tabellen als volgt gebruiken: A$ is voor de namen van de artikelen, A is voor de nummers van de artikelen en V is voor de voorraad.

Op de functie van regel 50 komen we direct terug. Op regel 60 staat een KOP-regel voor het voorraadoverzicht. Misschien blijkt straks, dat de indeling nog wat aangepast moet worden.

Omdat dit programma aanzienlijk groter wordt dan de programma's die we tot nu toe gemaakt hebben, willen we ook meteen voor overzichtelijkheid zorgen. Daarom nemen we niet alleen REM-opdrachten met toelichting op, maar kiezen we ook de regelnummers een beetje systematisch. Na de voorbereidingen gaan we niet direct door met het volgende regelnummer.
We nemen een "rond getal" voor het begin van het hoofdprogramma:

```
100 REM **HOOFDPROGRAMMA**
110 REM **INLEZEN TABELLEN**
120 FOR I=1 TO AA
130 READ A(I),A$(I)
140 IF A(I)<>0 THEN NEXT ELSE L=I-1
```

Eerst worden de gegevens van de dataregels overgebracht naar de tabellen A (artikelnummer) en A$ (omschrijving). Regel 140 laat zien dat de reeks gegevens op de dataregels afgesloten moet worden met een artikelnummer 0 waarbij wel een "artikelomschrijving" hoort. Artikelnummer 0 sluit af en telt dus zelf als artikelnummer niet mee. Als de MSX artikelnummer 0 ontdekt heeft legt hij in de variabele L vast hoeveel nummers en omschrijvingen in de tabellen staan.

Het aardige van het programma in BASIC is, dat we alles wat we gemaakt hebben meteen kunnen uitproberen.
Geef een RUN-opdracht.

We zien nu nog niet of het programma echt goed is. Daarvoor gebeurt er nog te weinig. Wat we wel zeker weten is, dat er in de opdrachten die de MSX tot nu toe heeft uitgevoerd, géén syntax errors zitten (of wel...? Dan moet er verbeterd worden!).

We kunnen er zelfs voor zorgen dat het programma, zoals we het nu hebben, helemaal goed "loopt" door voorlopig even wat "data" toe te voegen. Dat zijn nog niet de "echte" data; ze dienen alleen om te kunnen "testen" of het programma werkt.

Voeg daarom toe:

```
5000 DATA 34,voetbal,49,handbal,32,volleybal,18,tennisbal
5010 DATA 54,pingpongbal
5999 data 0,STOP
```

We zetten deze "data-paren" (artikelnummer en omschrijving) ver weg achter het programma. Er is dan ruimte genoeg om alles wat nog geprogrammeerd moet worden erbij te maken. Het is altijd handig om de afsluitende gegevens (hier 0 en STOP) alvast op een hoog regelnummer apart te zetten. Hier is dat gedaan op regel 5999. We hoeven dan later, als we de data echt gaan invullen, niets weg te halen. De afsluitende gegevens blijven staan en wij kunnen ze niet meer vergeten.

Geef nu weer een RUN-opdracht. Het programma moet met Ok, zonder verdere berichten, eindigen.

Vervolgens maken we het "menu", de lijst van keuzemogelijkheden:

```
200 REM **MENU**
210 CLS
220 PRINT "VOORRAADPROGRAMMA":PRINT "Keuze (status is " S$ "):":PRINT
230 PRINT"1=inlezen voorraadtabel (N,G)"
240 PRINT"2=overzicht voorraad (L,G)"
250 PRINT"3=bijwerken voorraad (L)"
260 PRINT"4=wijzigen voorraadtabel (L)"
270 PRINT"5=nieuwe tabel maken (N,G)"
280 PRINT"6=terugschrijven voorraadtabel (L)"
290 PRINT"9=stop programma (N,G)"
300 PRINT
310 INPUT "keuze";K
320 IF K < 1 OR K > 9 THEN PRINT "keuze fout, herhaal":GOTO 310
330 ON K GOTO 400,500,700,900,1100,1300,340,340,1500
340 K=0:GOTO 320
```

We hebben in het programma een vrije keuze uit zeven mogelijkheden. Sommige mogelijkheden zijn alleen zinvol onder bepaalde voorwaarden. Zo is het duidelijk dat we alleen de voorraad bij kunnen werken als we eerst een bestand gelezen hebben. Ook mogen we bijvoorbeeld niet stoppen met de uitvoering van het programma als de tabel nog niet naar de cassette teruggeschreven is.

Daarom hebben we het begrip STATUS ingevoerd. Daarvoor gebruiken we de variabele S$, die op regel 40 staat. In deze variabele tekenen we met een letter aan wat de status is van het programma. Een letter N betekent dat we nog geen bestand gelezen hebben. Bij deze status zijn de keuzemogelijkheden beperkt: alleen 5 en 9 komen in aanmerking. Achter elke keuzemogelijkheid is tussen haakjes aangegeven bij welke status de keuzemogelijkheid past.

Op regel 320 sluiten we de keuzemogelijkheden onder de 1 en boven de 9 uit. De keuzemogelijkheden 7 en 8, die er ook niet zijn, maar die we voor eventuele uitbreiding van het programma vrij gehouden hebben, worden hier niet uitgesloten. Daardoor kunnen we na het typen van de keuze gebruikmaken van een nieuwe opdracht.

Deze nieuwe opdracht is een bijzondere vorm van de opdracht GOTO. In het programma staat hij op regel 330. De opdracht zegt dat we naar één van de opgegeven programmaregels springen, afhankelijk van de waarde van de variabele K. Als K één is gaan we naar het eerste regelnummer, als K twee is naar het tweede enz. tot en met K=9, dan gaan we naar het negende regelnummer.

Als we 7 of 8 typen voor K is dit een foutieve keuze. Om de tekst die op het scherm moet komen bij een foutieve keuze niet nog eens in het programma te hoeven zetten maken we op regel 340 K gewoon gelijk aan 0 en springen terug naar regel 320. Daar blijkt K dan een ongeldige keuze te zijn.

We kunnen het maken van de keuze nog iets mooier en veiliger programmeren, op deze manier:

```
310 PRINT "Keuze: ";
320 K$=INPUT$(1):K=INSTR("1234569"):IF K=0 THEN 320 ELSE PRINT K
330 ON K GOTO 400,500,700,900,1100,1300,1500
```

Op regel 310 staat de mededeling dat er een keuze gemaakt moet worden. Op regel 320 is het invoeren van die keuze geregeld met een INPUT$-opdracht voor één teken. RETURN hoeft dan niet meer. Dat ene teken komt in K$ te staan. Vervolgens kijken we op welke plaats in de string "1234569" de getypte K$ voorkomt. De string "1234569" bevat alle tekens die gekozen mogen worden. De uitslag van de INSTR-opdracht (de plaats dus waar K$ in "1234569" zit) komt in K.

Als K=0, dan zit de gekozen toets niet in het rijtje "1234569". Dat betekent niets meer of minder dan: er is iets gekozen dat helemaal niet gekozen mag worden. In dat geval gaan we terug naar het begin van regel 320 en vragen een andere toets. We zetten niets op het scherm, zodat de tekst op het scherm leesbaar blijft, hoe vaak er ook iets verkeerds gekozen wordt (dat was in de oude oplossing anders!).

Als K niet 0 is, dan moet hij 1, 2, 3, 4, 5, 6 of 7 (niet 9!) zijn. K geeft dan aan: uit het rijtje "1234569" is de K-de gekozen. Daarmee kunnen we direct naar de ON K GOTO-opdracht gaan. Bij de eerste keuze (cijfer 1) hoort regelnummer 400. Bij de tweede keuze (cijfer 2) hoort regelnummer 500. Bij de zevende keuze (met cijfer 9) hoort het zevende regelnummer: 1500.

Opmerking: op regel 330 kunnen we, zolang we nog niet precies weten hoe groot elk programmadeel wordt, zo maar wat getallen invullen. Het gemakkelijkste is het om voorlopig de nummers van de keuzemogelijkheden te gebruiken. We kunnen dan na elk programmadeel dat we aan het programma toevoegen de beginregel in deze opdracht vermelden en gemakkelijk zien waar die moet staan. Hier dus:

```
330 ON K GOTO 1,2,3,4,5,6,9
```

Als we eenmaal zo ver zijn kunnen we de delen van het programma gaan maken. We hoeven dat niet persé te doen in de volgorde van de keuzelijst. We kunnen ook eerst de gemakkelijkste delen afhandelen en later de moeilijkste doen of omgekeerd. In dit voorbeeld houden we ons gewoon aan de volgorde van de keuzelijst.

110. De uitwerking

We willen beginnen met keuzemogelijkheid 1, het inlezen van een bestand in de voorraadtabel V. Er is echter een probleem: als we dat gaan uitwerken kunnen we het niet proberen, omdat we nog geen bestand hebben. Bovendien: het uitproberen met de cassette houdt het werk zo op.

Daarom gaan we voorlopig even smokkelen: we doen net of we een bestand lezen, maar intussen grabbelen we zomaar wat getallen bij elkaar:

```
400 REM**inlezen tabel**  (voorlopig)
410 IF S$="L" THEN PRINT "Tabel reeds aanwezig":GOTO 310
420 PRINT "Bestand lezen (j/n) ";:JN$=INPUT$(1):PRINT
430 IF JN$<>"j" THEN S$="N":GOTO 200
440 FOR I=1 TO L:V(I)=INT(RND(5)*40):NEXT I
450 S$="L":GOTO 200
```

We mogen geen nieuwe tabel lezen als we in het programma al een tabel gelezen hebben die nog niet teruggeschreven is naar de cassette. In het programma betekent dat dus dat we teruggaan naar de keuzelijst als in S$ de status van "al een cassette gelezen" staat. Dat gebeurt op regel 410. Daarna zetten we een tekst op het scherm die aangeeft dat er een cassette in de cassetterecorder gezet moet worden. We weten dat het belangrijk is deze tekst in het programma op te nemen. De INPUT$-opdracht geeft ons bovendien de mogelijkheid ons nog te bedenken en bij nader inzien toch maar geen bestand te lezen. Ook dan keren we terug naar de keuzelijst.

Als we met de kleine letter j antwoorden op de vraag, gaat de MSX de tabel V voor L artikelen vullen met een "schijnvoorraad". Het lijkt dan of we een bestand gelezen hebben, waarna de MSX de status verandert in L: er is een cassette gelezen. Daarna keren we terug naar de keuzelijst. Dit stukje programma kan rustig geprobeerd worden! Als alles werkt, moet in het menu ná keuze 1 de status veranderd zijn. Kiezen we dan nog een keer 1, dan moet de MSX melden: "Tabel reeds aanwezig".

Keuzemogelijkheid 2 is het maken van een overzicht van de voorraad op het scherm.

```
500 REM **VOORRAADOVERZICHT**
510 IF S$="N" THEN PRINT "Geen tabel gelezen":GOTO 310
520 R=1
530 FOR I=1 TO L
540 IF R=1 THEN GOSUB 600
550 PRINT A(I) TAB(10) A$(I) TAB(34) V(I)
560 R=R+1:IF R>18 THEN PRINT "Doorgaan met een toets ":X$=INPUT$(1)
570 NEXT
580 LOCATE 0,22
590 PRINT "Einde overzicht; doorgaan met een toets ":X$=INPUT$(1):GOTO 200
600 CLS: PRINT KOP$:PRINT STRING$(40,"_"):RETURN
```

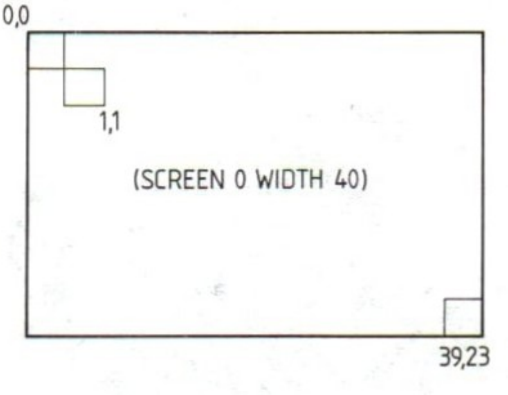
In dit programmadeel zitten twee bijzonderheden die we willen bespreken.
In de eerste plaats is dat de oplossing van het volgende probleem: als we een overzicht maken van de voorraad van de artikelen op het scherm weten we dat dat overzicht niet in één keer op het scherm past als er meer dan 24 artikelen zijn. Als we bovendien een KOPregel op het scherm willen hebben kunnen we hoogstens 23 artikelen op het scherm kwijt. Als in onze tabel meer dan 23 artikelen staan krijgen we dus het probleem dat de tekst op het scherm naar boven gaat schuiven zodra we bij het 24e artikel aangekomen zijn. De KOPregel is dan verdwenen en bovendien verdwijnen de gegevens van de artikelen bovenaan in de tabel.

We hebben er daarom voor gezorgd dat de MSX steeds hooguit 18 artikelen op het scherm zet. Hij stopt dan en wacht onze opdracht af voordat hij de volgende 18 op het scherm plaatst.
Om dit voor elkaar te krijgen hebben we een variabele R ingevoerd waarin we regels tellen. Telkens als R boven de 18 komt laten we de MSX stoppen en ons antwoord afwachten voordat hij doorgaat. R begint dan weer bij 1 te tellen.

Bij deze oplossing wordt bewezen hoe handig het gebruik van subroutines is: elke keer als het scherm "vol" is en we weer bij regel 1 willen beginnen, gaan we "even langs" bij 600. Daar wordt het scherm schoongemaakt, een kopregel op het scherm gezet en we kunnen weer terug naar de plaats vanwaar we gekomen zijn met de opdracht RETURN **(niet de toets RETURN).**

De tweede bijzonderheid staat op regel 580: de opdracht LOCATE. Met deze opdracht kunnen we de cursor naar een bepaalde plaats op het scherm sturen.

In de opdracht LOCATE staat voorop het positienummer, daarachter een komma en daarachter het regelnummer. Zowel de posities als de regels beginnen met 0 te tellen (net als bij b.v. TAB).



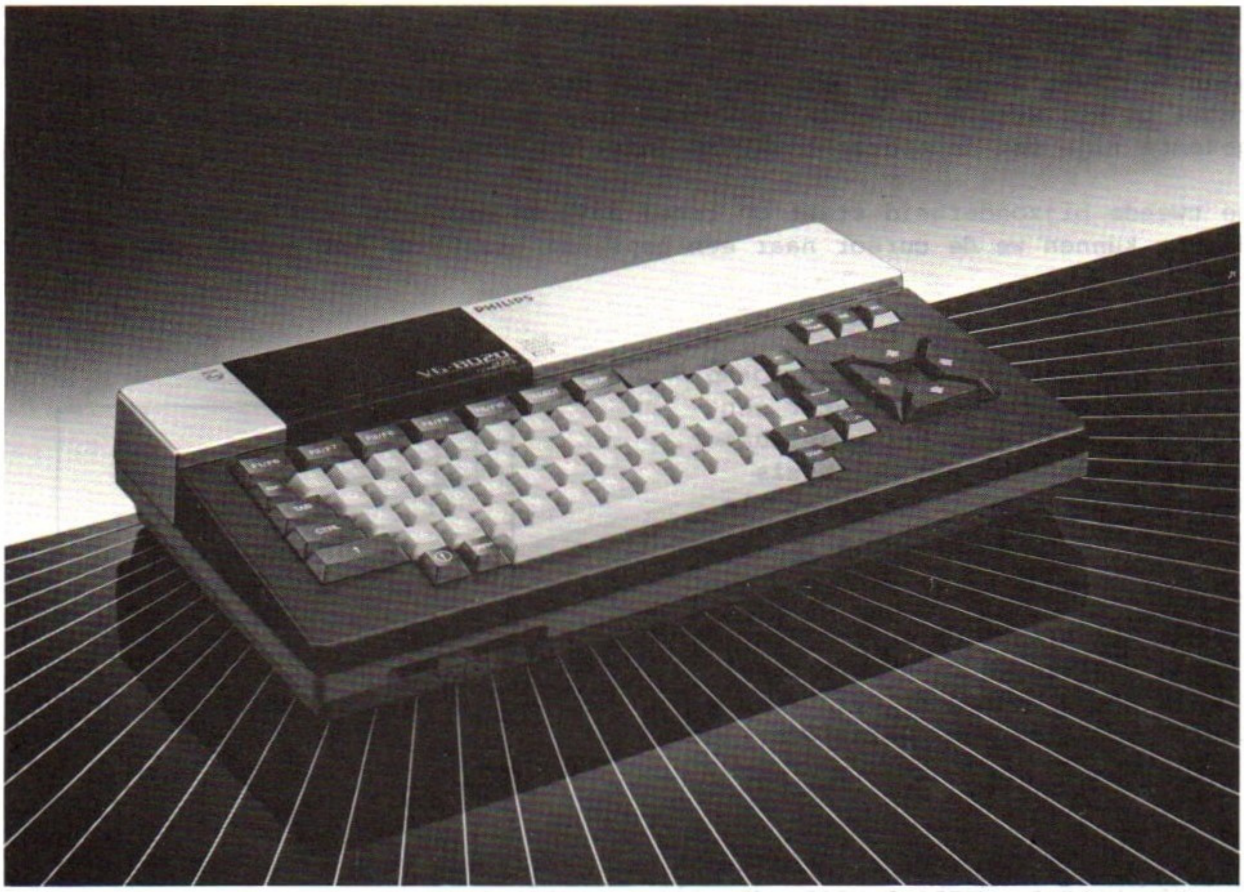
Dus:
LOCATE 1,1 brengt de cursor naar de tweede positie op de tweede regel.
LOCATE 29,23 brengt de cursor naar de laatste positie van de laatste regel.

De getallen mogen weer als constante, als variabele of als berekening weergegeven worden. Probeer maar even als tussendoortje (op een schoon scherm:

P=39:FOR R=23 TO 0 STEP -1:P=P-1:LOCATE P,R:PRINT"*";:NEXT

Op regel 590 staat weer een INPUT$-opdracht die geen andere taak heeft dan ervoor te zorgen dat de MSX niet direct doorgaat, maar wacht op een seintje van ons. Deze opdracht moet er staan omdat de MSX anders na het laatste deel van het overzicht onmiddellijk het scherm schoon zou maken voor de lijst met keuzemogelijkheden. We zouden dan geen gelegenheid hebben het laatste deel van het overzicht te lezen.

We kunnen nu twee delen van het programma testen: keuze 1 en keuze 2. Denk eraan dat keuze 2 pas mag als keuze 1 een keer uitgevoerd is.



111. Meer delen

Keuzemogelijkheid 3 betreft het bijwerken van de tabel met voorraden. In dit deel van het programma is het de bedoeling dat we de voorraad kunnen verhogen (bij) of verlagen (af). Dit deel mag alleen uitgevoerd worden als de tabel gelezen is van de cassette.

In dit deel van het programma kunnen we meer dan één, of als we willen zelfs alle voorraden bijwerken. We gaan dan niet steeds terug naar de keuzelijst. Daarom hebben we in dit programmadeel de mogelijkheid ingebouwd om óf door te gaan met het bijwerken van de voorraad dan wel te stoppen met dit programmadeel en terug te keren naar de keuzelijst. We laten dat afhangen van het artikelnummer dat getypt wordt. Artikel nummer 0 bestaat niet, dus kunnen we 0 gebruiken als signaal om op te houden met bijwerken.

```
700 REM **BIJWERKEN TABEL**
710 IF S$<>"L" THEN PRINT "Geen tabel gelezen":GOTO 310
720 CLS
730 INPUT "Artikelnummer (stop met 0)";AN
740 IF AN<1 THEN 200
750 FOR I=1 TO L
760 IF A(I)=AN THEN 780 ELSE NEXT
770 PRINT "Nummer bestaat niet":PRINT:GOTO 730
780 PRINT "Artikel" A(I) A$(I)
790 PRINT "Voorraad is" V(I);:INPUT "bij of af";M
800 V(I)=V(I)+M:PRINT "Voorraad nu" V(I):PRINT:GOTO 730
```

De MSX zoekt of het artikelnummer dat wij typen in de tabel staat. Vindt hij het, dan zet hij de omschrijving van het artikel erachter en geeft aan hoe de voorraad nu is. Vindt hij het niet, dan krijgen we daarvan een bericht. Als de artikelomschrijving en de voorraad nu op het scherm staan kunnen we een positief getal typen om iets bij de voorraad te doen of een negatief getal om iets van de voorraad af te halen.

In het vierde programmadeel kunnen we de voorraad wijzigen:

```
900 REM **WIJZIGEN TABEL**
910 IF S$<>"L" THEN PRINT "Geen tabel gelezen":GOTO 310
920 CLS
930 INPUT "Wijzig artikelnummer (0=stop)";AN
940 IF AN<1 THEN 200
950 FOR I=1 TO L
960 IF A(I)=AN THEN 980 ELSE NEXT
970 PRINT "Nummer bestaat niet":PRINT:GOTO 930
980 PRINT "Artikel" A(I) A$(I)
990 PRINT "Voorraad was" V(I) "wordt";
1000 INPUT V(I):PRINT:GOTO 930
```

Wijzigen is iets anders dan bijwerken. Bij het bijwerken doen we bij de bestaande voorraad iets bij of we halen er iets af. Bij het wijzigen vervangen we de voorraad die in de tabel staat door een andere. Fouten kunnen een reden zijn om wijzigingen in de tabel aan te brengen.

Dit programmadeel kan ook gebruikt worden als we aan een bestaande tabel een artikel willen toevoegen. We voegen dan eerst het nummer en de omschrijving toe in een dataregel (vóór artikelnummer 0 met omschrijving STOP op regel 5999). Daarna moeten we zorgen dat de beginvoorraad van dit artikel in de tabel komt.
We lezen dan de tabel in, kiezen voor wijzigen van de tabel, typen het nummer van het nieuwe artikel en kunnen dan de voorraad wijzigen in de beginvoorraad.
Datzelfde doen we helemaal aan het begin, dat wil dus zeggen als we nog geen tabel met artikelvoorraden hebben. Wijzigen kunnen we natuurlijk alleen in een gelezen tabel. De status in S$ moet dus L zijn.

We komen nu aan keuzemogelijkheid 5: het maken van een nieuwe tabel. Dit deel is in het programma opgenomen omdat we de eerste keer dat we het programma uitvoeren nog geen tabel op een cassette hebben staan. We kunnen dus ook geen tabel lezen om die vervolgens te wijzigen of bij te werken.

Door dit programmadeel te kiezen zeggen we tegen de MSX: maak een nieuwe tabel V met in alle elementen het getal 0.

```
1100 REM **NIEUWE TABEL**
1110 IF S$<>"G" AND S$<>"N" THEN PRINT "Nog tabel aanwezig":GOTO 310
1120 PRINT "Wacht"
1130 FOR I=1 TO AA:V(I)=0:NEXT
1140 PRINT "Tabel gereed":S$="L":GOTO 310
```

Opmerking: In de meeste gevallen wordt deze functie uit het programma gelaten en maken we daar een apart programma of programmaatje voor. Dat is wel zo handig omdat dit programmadeel eigenlijk maar één keer nodig is, n.l. bij het voor de allereerste keer maken van een tabel. Zo'n apart programma dat niets anders te doen heeft dan een lege tabel op de cassette te zetten, noemen we dan een "aanmaak-programma".

Het schrijven van de gewijzigde, bijgewerkte of nieuwe tabel op de cassette doen we met het volgende programmadeel:

```
1300 REM **BESTAND SCHRIJVEN**
1310 IF S$<>"L" THEN PRINT "Tabel al geschreven":GOTO 310
1320 PRINT "Plaats cassette, start (j/n) ":JN$=INPUT$(1):PRINT
1330 IF JN$<>"j" THEN 200
1340 REM  hier moet schrijven komen
1350 PRINT "Cassette klaar; kopie maken":GOTO 1320
```

Ook hier geven we niet onmiddellijk de schrijfopdracht voor de cassette omdat we even willen waarschuwen dat er een cassette (met een stopje) in de recorder gezet moet worden. Als dat vergeten wordt stopt het programma zonder te schrijven.

Op regel 1350 springen we terug naar regel 1320. We krijgen daardoor de mogelijkheid de tabel ook nog op een tweede cassette te schrijven, zodat we een kopie van de tabel kunnen maken. Dat is zowel voor programma's als voor bestanden op cassette van belang: er kan altijd iets misgaan met een cassette waardoor we het programma of het bestand kwijtraken. De copie moet natuurlijk wel op een andere cassette komen.

Opmerking: het "echte" schrijven hebben we nog even achterwege gelaten. Regel 1340 is daarvoor vrijgehouden.

Tenslotte hebben we dan nog het stoppen van het programma:

```
1500 REM **STOP**
1510 IF S$="L" THEN PRINT "Tabel niet geschreven":GOTO 310
1520 CLS
1530 INPUT "Start programma (naam)";P$
1540 CLOAD P$
```

Stoppen mag niet als er nog een tabel in het geheugen zit die niet teruggeschreven is naar de cassette.
In dit programmadeel hebben we nog iets bijzonders gedaan: op regel 1530 kunnen we de naam ingeven van het volgende programma dat uitgevoerd moet worden. In de CLOAD op 1540 staat nu geen programmanaam maar een stringvariabele waarin de programmanaam te vinden is.

Het programma is nu compleet op de stukjes na die moeten gaan zorgen voor het "echte" lezen en schrijven van het bestand op de cassette. Daar kijken we in de volgende paragraaf naar.

Schrijf het volledige programma op cassette (liefst met een copie op een tweede cassette). Test het daarna.
Testen wil zeggen: probeer het helemaal uit. Laat de MSX alle keuzemogelijkheden doorlopen. Kijk of het scherm er netjes uitziet. Probeer grote en kleine getallen. Kijk wat er gebeurt als we iets anders typen dan er gevraagd wordt.
Pas als we over het programma tevreden zijn, gaan we het in de volgende paragraaf afmaken.

112. De bestandsroutines

Tot slot moeten we het lezen en schrijven van het bestand in het programma verwerken. In feite liggen deze stukken programma al klaar in de paragrafen aan het begin van dit hoofdstuk. We hoeven ze slechts in te passen in het programma. Dat gaat altijd het gemakkelijkst met subroutines.

De plaatsen waar het lezen en schrijven in het programma moeten plaatsvinden zijn al bekend:

```
440 GOSUB 2000:REM bestand lezen
1340 GOSUB 3000:REM bestand schrijven
```

Op regel 440 vervangen we de tijdelijke oplossing nu door de definitieve: we gaan even langs bij 2000 om een bestand te lezen.
Op regel 1340 doen we hetzelfde voor het schrijven van het bestand.

We werken de twee subroutines uit zoals we in de eerste paragrafen van dit hoofdstuk hebben gedaan:

```
2000 REM **leesroutine**
2010 TX$="recorder op weergeven":GOSUB 4000
2020 I=1
2030 OPEN "CAS:voorra" FOR INPUT AS #1
2040 IF EOF(1) THEN 2080
2050 INPUT #1,V(I)
2060 I=I+1
2070 GOTO 2040
2080 CLOSE #1
2090 TX$="stop recorder":GOSUB 4000
2100 RETURN
```

In dit leesprogramma hebben we opnieuw een subroutine met GOSUB gebruikt. Twee keer zelfs: op regel 2010 en op regel 2090. Dit is een typisch voorbeeld van het gebruik van een subroutine. Elke keer als we een bericht op het scherm willen hebben, dat alleen met een toetsindruk beantwoord behoeft te worden, moeten we hetzelfde stukje programma schrijven. Dat komt dus op verschillende plaatsen in het programma voor. Het is daarom veel gemakkelijker om van dit stukje programma een subroutine te maken.
Het stukje programma is echter niet elke keer <u>precies</u> hetzelfde. De tekst van het bericht verschilt. De oplossing is dan simpel: we laten die tekst uit de "geef-een-bericht-en-wacht-op-een-toets-routine" weg en "geven die tekst mee" aan de routine in de vorm van een variabele:

```
2010 TX$="recorder op weergeven":GOSUB 4000
2090 TX$="stop recorder":GOSUB 4000
```

De routine wordt dan:

```
4000 REM **berichtroutine**
4010 LOCATE 0,22
4020 PRINT TX$ " (toets) ";
4030 X$=INPUT$(1)
4040 LOCATE 0,22
4050 PRINT SPC(40)
4060 RETURN
```

Deze routine zet een bericht TX$ op regel 22, positie 0 (dankzij LOCATE). Achter dit bericht komt de tekst "(toets)" te staan. In deze tekst is rekening gehouden met de spaties. Op regel 4030 wordt vervolgens gewacht tot er een toets ingedrukt wordt.
Op regel 4050 worden 40 spaties op het scherm gezet. De functie SPC kan alleen in een PRINT-opdracht gegeven worden. Het getal tussen haakjes geeft het aantal spaties aan. Deze spaties zorgen ervoor dat het bericht van het scherm gewist wordt, ook weer dankzij de LOCATE-opdracht. Het "weghalen" van het bericht is belangrijk: als we bij een INPUT-opdracht tijdens de uitvoering van het programma doen wat er gevraagd wordt, moet de opdracht om dat te doen ook onmiddellijk van het scherm verdwijnen.
De opdracht RETURN zorgt er vervolgens voor dat we weer terugkeren naar de plaats waar we vandaan zijn gekomen.

Dezelfde "bericht-routine" gebruiken we in de schrijfroutine:

```
3000 REM **schrijfroutine**
3010 TX$="recorder op weergeven":GOSUB 4000
3020 OPEN "CAS:voorra" FOR OUTPUT AS #1
3030 FOR I=1 TO AA
3040 PRINT #1,V(I);
3050 NEXT I
3060 CLOSE #1
3070 TX$="stop recorder":GOSUB 4000
3080 RETURN
```

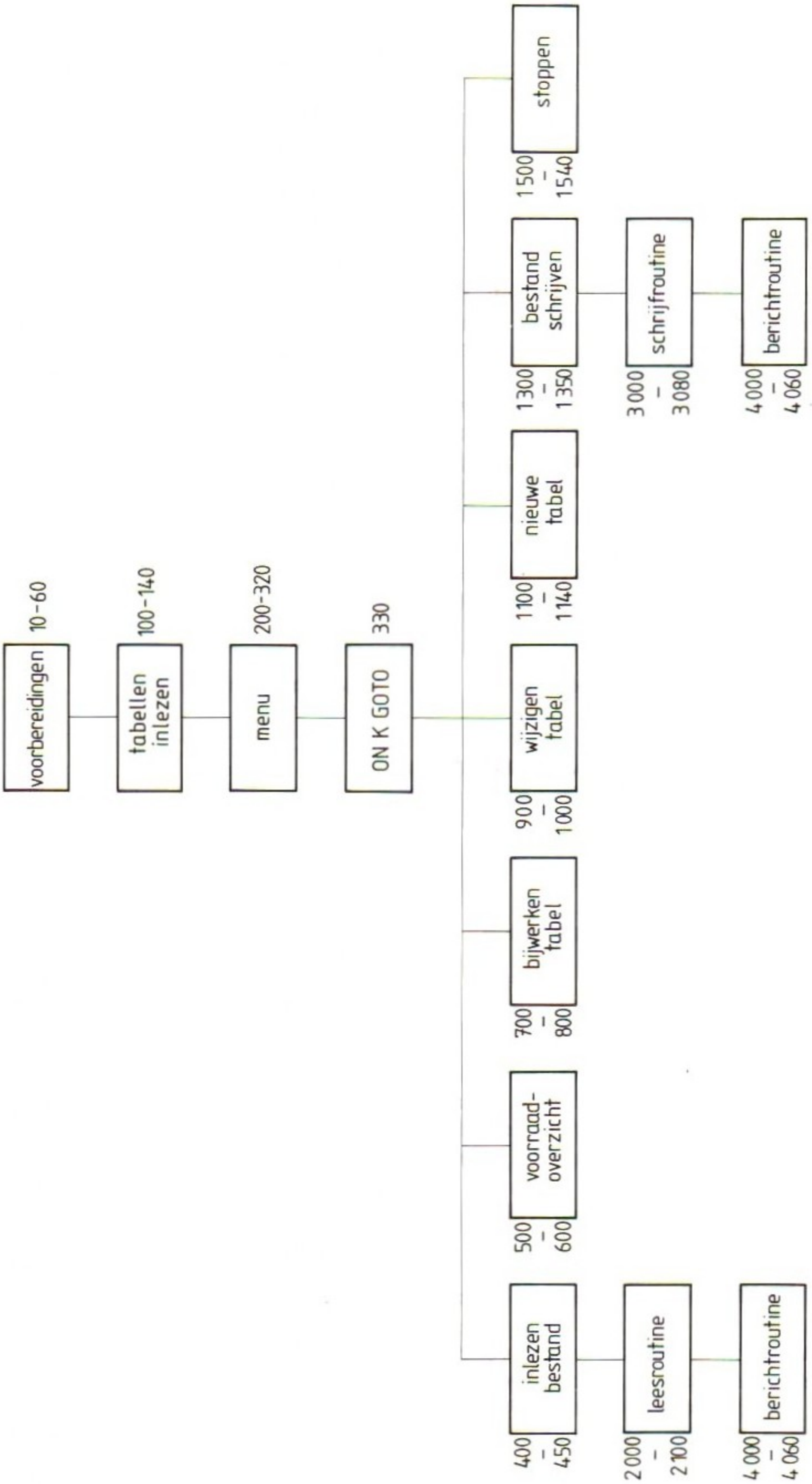
En daarmee is het programma compleet.

Bij grotere programma's is het altijd nodig de structuur van het totaal overzichtelijk te houden. Daarom eindigen we dit hoofdstuk met een schema van het voorraadprogramma dat we in dit hoofdstuk gemaakt hebben. Voor de duidelijkheid hebben we de regelnummers bij de onderdelen van het programma gezet.

Voor het overzichtelijk houden van een programma is één ding van het grootste belang:

Gebruik nooit GOTO-opdrachten van het ene onderdeel naar het andere.

In het schema zijn de onderdelen voorgesteld met vakjes. We hebben ons netjes aan deze regel gehouden: binnen een vakje hebben we wel GOTO-opdrachten gebruikt, maar nooit om van het ene vak in het andere te komen. De weg leidt altijd netjes langs de "centrale verdeelopdracht", de ON K GOTO.



voorbereidingen
10-60
tabellen inlezen
100-140
menu
200-320
ON K GOTO
330
inlezen bestand
400 - 450
leesroutine
2000 - 2100
berichtroutine
4000 - 4060
voorraad-overzicht
500 - 600
bijwerken tabel
700 - 800
wijzigen tabel
900 - 1000
nieuwe tabel
1100 - 1140
bestand schrijven
1300 - 1350
schrijfroutine
3000 - 3080
berichtroutine
4000 - 4060
stoppen
1500 - 1540

HOOFDSTUK 14: Muziek en grafiek

In dit slothoofdstuk staan een aantal betrekkelijk korte probeersels met opdrachten, die we nog niet gebruikt hebben. We doen o.a. iets aan muziek en aan de grafische mogelijkheden van de MSX.

De paragrafen over deze onderwerpen zijn in dit probeerboek bedoeld als een kennismaking: beide onderwerpen zijn zo omvangrijk en bieden zoveel mogelijkheden, dat een uitvoerige bespreking niet in dit boek past.

113. Kortjakje

Probeer:

```
10 CLEAR 1000:CLS
20 A$=INPUT$(1)
30 PRINT A$;
40 PLAY A$
50 GOTO 20
```

Voer het programma uit en typ de letters van een toonladder: cdefgabc. Zet het geluid van de tv voldoende hard!

De opdracht PLAY speelt de tonen van de toonladder. Deze opdracht vraagt om een toon of een melodie in de vorm van een string (als constante, variabele of "berekening").
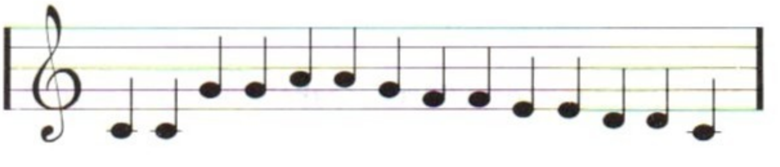
Wijzig regel 20 en 30:

```
20 INPUT "melodie: ";A$
30 PRINT A$
```

Nu kunnen we meer tonen invoeren. Luister naar het resultaat.

Op letters die niet in de toonladder thuishoren, reageert de MSX met "Illegal function call". Dat is echter niet met alle letters die geen toon voorstellen het geval. Sommige letters hebben namelijk een stuurfunctie. Ze zorgen ervoor dat de melodie anders gaat klinken.

```
10 CLEAR 1000:CLS
20 A$="ccggaagffeeddc"
30 PLAY A$
```



Dit melodietje lijkt al een beetje op Kortjakje, maar de MSX maakt geen duidelijk verschil tussen twee gelijke tonen na elkaar: cc klinkt als één toon c.

Daar brengen we verandering in door de melodie anders te sturen:

20 A$="S8M9000ccggaagffeeddc"

Alle tonen klinken nu "los van elkaar". De combinatie van stuurtekens S8 en M9000 zorgt daarvoor. In het BASIC-boekje bij de computer is te lezen, welke waarden S en M kunnen krijgen. Merk op dat de stuurtekens en de waarden die erbij horen in de string staan. Merk ook op dat de computer alweer Ok zegt, voordat hij het hele melodietje gespeeld heeft!

We verfraaien nu het lied met een "rust":

20 A$="S8M9000ccggaagR4ffeeddc"

In dit lied klinkt "midden in de week" niet goed: er zouden 4 korte tonen f moeten klinken. Dat kan zo:

20 A$="S8M9000ccggaagR4L8ffffL4eeddc"

De stuurletter L bepaalt de lengte van de volgende tonen. Maak het getal achter L tweemaal zo groot, dan wordt de lengte van de toon half maal zo lang.

Opmerking: alle stuurmogelijkheden hebben een beginwaarde die de MSX aanhoudt, zolang we hem niet wijzigen. Voor de L is die waarde 4.
De computer kan de melodie ook hoger of lager spelen (let op het verschil: Ø is nul en O is Oh):

20 A$="O6S8M9ØØØccggaagR4L8ffffL4eeddc"

De beginstand is O4. Een hoger getal geeft dus een hogere toon. Met de T bepalen we het tempo. De beginwaarde is T120. Probeer:

20 A$="T2ØØO4S8M9ØØØccggaagR4L8ffffL4eeddc"

De vier toontjes f zitten nu weer bijna aan elkaar geplakt. Dat kunnen we veranderen door de waarde achter M te verlagen. Maak ervan:

20 A$="T2ØØO4S8M5ØØØccggaagR4L8ffffL4eeddc"

De letter V bepaalt het volume. De beginstand is 4, het hoogste volume is 15. Verander regel 20:

20 A$="V15T2ØØO4S8M5ØØØccggaagR4L8V8ffffL4eeddc"

Door de verandering van het volume midden in de melodie gaat de werking van de stuurtekens S en M weer verloren. Dat herstellen we zo:

20 A$="V15T2ØØO4S8M5ØØØccggaagR4L8V8S8M5ØØØ8ffffL4eeddc"

Er valt dus heel wat te sturen en te regelen bij het muziek maken!

We gaan nu een tweede en een derde stem aan Kortjakje toevoegen. Gebruik de edit-mogelijkheden om regel 20 twee keer te copiëren (naar regel 20, regelnummer veranderen in 22, naar vrije regel, list, naar regel 22, veranderen in 24; vergeet de RETURN niet!).

Verander daarna de nieuwe regels:

22 B$= (laat de rest even staan)
24 C$= (laat de rest staan)
30 PLAY A$,B$

De derde stem staat al wel klaar. maar wordt nog niet gespeeld.
Verander op regel 22 de toonladder 06 in 04:

22 B$="V15T2ØØ04 (en de rest laten we zo)

Dat klinkt al anders. Maak nu van regel 22 een tweede stem:

22 B$="V15T2ØØO4S8M5ØØØcceeffeR4L8V8S8M5ØØØccccL4ccbbc"

De laatste tonen van de tweede stem klinken te hoog: ze moeten een octaaf lager gespeeld worden:

22 B$="V15T2ØØO4S8M5ØØØcceeffeR4L8V8S8M5ØØØccccL4ccO3bbO4c"

Opmerking: in de string mogen zowel hoofd- als kleine letters gebruikt worden. Dat maakt noch voor de tonen, noch voor de stuurtekens enig verschil. Spaties hebben geen functie: ze kunnen gebruikt worden om het geheel overzichtelijker te maken.

Maak nu zelf iets moois van de derde stem.

114. Experimenteren met PLAY

Omdat de PLAY-opdracht met strings werkt, is het proberen van al die stuurmogelijkheden wat lastig. Gelukkig heeft MSX-BASIC daarvoor een voorziening: de waarden bij de stuurtekens mogen in de vorm van een variabele binnen de string opgegeven worden in de volgende vorm:

```
10 CLEAR 1000:CLS
20 FOR I=1 TO 8
30 PRINT I;
40 PLAY "O=I;cedfgab"
50 X$=INPUT$(1)
60 NEXT I
```

De nieuwe mogelijkheid staat op regel 40. O is de O van octaaf. Zowel het =-teken als de puntkomma zijn belangrijk. Het programma laat de toonladder op alle 8 toonhoogten horen.

Opmerking: de kunstmatige onderbreking is in het programma opgenomen, zodat we kunnen zien welk geluid bij de waarde van 0 hoort. Anders PRINT de MSX sneller dan hij speelt en kunnen we het nog niet goed volgen. RESET de computer voordat aan dit nieuwe programma wordt begonnen. We krijgen dan van alle stuurtekens de beginwaarden weer.

Probeer op dezelfde manier:

```
20 FOR I=32 TO 255 STEP 20
40 PLAY "T=I;cedfgab"
```

Probeer ook V (van 0 t.e.m. 15), S (van 0 t.e.m. 15) en M (van 1 t.e.m. 65535 met flinke stappen).

Tenslotte een programmaatje om de samenhang tussen S en M hoorbaar te maken:

```
10 CLEAR 1000:CLS
20 S(1)=1:S(2)=4:S(3)=8:S(4)=10:S(5)=11:S(6)=12:S(7)=13:S(8)=14
30 M(1)=255:M(2)=9000:M(3)=32000
40 L(1)=4:L(2)=32:L(3)=64
50 FOR S=1 TO 8
60 PRINT "S=" S(S) TAB(6) "M=" M(M);
70 FOR M=1 TO 3
80 PRINT TAB(4+M*6) M(M);
90 FOR L=1 TO 3
100 PLAY "V15;S=S(S);M=M(M);L=L(L);04 CEG 05 CEG 06 C"
110 NEXT L
120 X$=INPUT$(1)
130 NEXT M
140 PRINT
150 NEXT S
160 PRINT "Einde programma":END
```

Opmerkingen:

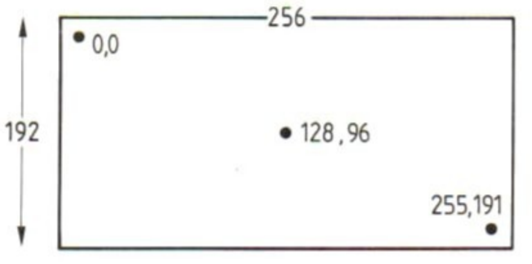
1. De af te werken waarden zijn in drie tabellen gezet. De andere waarden van S hebben hetzelfde effect als de waarden die in de tabel staan. De waarden van M en L zijn steeds "laag", "in het midden" en "hoog" gekozen.
2. In tegenstelling tot wat in sommige MSX-boekjes bij de machine staat, kan de waarde van M maximaal 32767 zijn. M mag echter wel negatief zijn!
3. In het programma is te zien dat naast tabel S, waarvan de elementen met een index worden aangeduid, ook nog een variabele S gebruikt mag worden. S(S) is dus het element uit variabele S, waarvan de index (het volgnummer) in de variabele S staat. Op regel 100 maken we het helemaal bont: S=S(S). De eerste S is een letter in de muziekstring, de tweede S is een tabelnaam, de derde een variabele-naam.
4. Let bij het experimenteren met muziek op voor de V in de string. Een verandering in het volume ergens in de string kan tot gevolg hebben dat het effect van andere stuurtekens weer verloren gaat. Zet de V dus altijd vooraan.

115. Grafieken

De MSX heeft veel, mooie grafische mogelijkheden. Er is echter één probleempje: anders dan bij het "gewone" programmeren en bij de muziekprobeersels, kunnen we de grafische mogelijkheden niet direct uitproberen maar uitsluitend via een programma.

De grafische mogelijkheden van de MSX zitten op SCREEN 2 en SCREEN 3, de schermsoorten die we al eerder ontmoet hebben. Zodra de MSX klaar is met een directe opdracht of met een programma, keert hij altijd onmiddellijk naar schermsoort 0 of 1 terug. De grafische probeersels verdwijnen dus onmiddellijk weer van het scherm. Op de grafische schermen kunnen we niet programmeren. We zullen het dus anders moeten doen.

```
10 CLEAR 1000:CLS
20 SCREEN 2
30 FOR R=0 TO 191 STEP 2
40 FOR P=0 TO 255 STEP 2
50 PSET (P,R)
60 NEXT P
70 NEXT R
80 X$=INPUT$(1)
```



Voer het programma uit en bekijk het resultaat: het scherm wordt volgeschreven met puntjes. SCREEN 2 heeft 192 regels, genummerd van 0 tot en met 191. Op elke regel passen 256 puntjes, genummerd van 0 tot en met 255. Met de opdracht PSET(P,R) wordt een puntje gezet op positie P van regel R.

In het probeerprogramma hebben we steeds één puntje overgeslagen, anders zou het scherm helemaal volgeschreven worden en zouden we het effect nog niet goed kunnen zien.

Het programma eindigt pas als we een toets indrukken (anders blijft SCREEN 2 aan het einde niet staan).

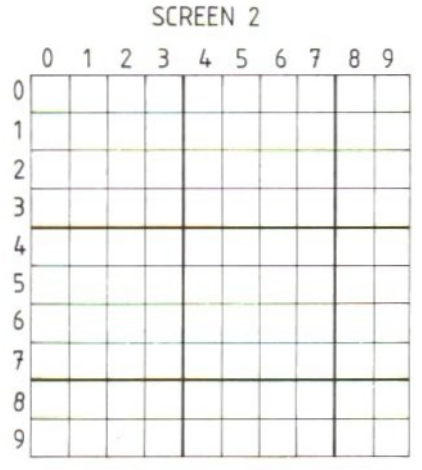
In de PSET-opdracht kunnen we ook een kleur meegeven voor de puntjes:

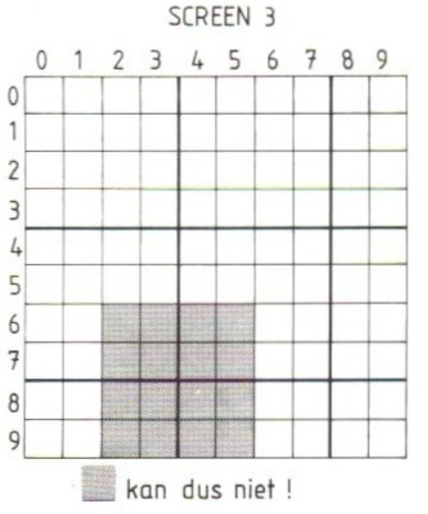
```
15 COLOR 15,14,1
50 PSET (P,R),6
```

Op het vierde scherm, SCREEN 3, zijn de puntjes groter. In feite zijn het blokjes van 4 bij 4 puntjes op het scherm van 192 bij 256 punten. Pas het programma aan:

```
20 SCREEN 3
30 FOR R=0 TO 191 STEP 8
40 FOR P=0 TO 255 STEP 8
```

Opmerking: een kleinere stap heeft geen zin; de MSX gaat op het scherm ook uit van grotere blokjes:





Dat kan zo geprobeerd worden:

40 P=P+1:REM FOR P= (enzovoorts; laat de rest van de regel staan)
60 REM NEXT

Met de REM-opdracht schakelen we de oude opdrachten even uit om het resultaat te bekijken. Straks gaan we weer terug naar het originele programma.

Bij de uitvoering van het programma zien we dat bovenstaande afbeelding goed is. Breng het programma weer in de oude vorm.

Elk blokje kan een eigen kleur krijgen. Probeer:

50 PSET(P,R),INT(RND(5)*16)

De kleuren kunnen zelfs direct naast elkaar staan. Verander de STEP op regel 30 en 40 maar in 4.

Kijk nu hoe dat er op SCREEN 2 uitziet met STEP 2 en STEP 1. De MSX heeft wel even wat tijd nodig om het hele scherm zo gedetailleerd vol te schrijven!

Op scherm 3 krijgt niet elk puntje een eigen kleur. Dat kán toeval zijn, maar het lijkt erop dat het dat niet is. Om zeker te zijn geven we de MSX opdracht elke punt een andere kleur te geven:

```
25 K=4
50 PSET(P,R),K
55 IF K=4 THEN K=6 ELSE K=4
```

We maken de kleur van de stippen afwisselend blauw (4) en rood (6). Die twee zijn op een licht scherm het beste te onderscheiden.

We zien bij de uitvoering van het programma dat alles nu blauw wordt. Bij STEP 2 op regel 40 wordt alles rood, bij STEP 3 komen er (eindelijk) afwisselend 4 blauwe en 4 rode stippen op het scherm.

Opmerking: de eerste blauwe stip kan wat kleurinvloed ondervinden van de laatste rode.

De kleur wordt bepaald per blokje van 4 stippen zoals op scherm 3.

Door het plaatsen van stippen kunnen we zelf een tekening maken. Stippen kunnen ook gewist worden, waardoor een "beweging" gesuggereerd kan worden. Probeer:

```
10 CLEAR 1000:CLS
20 COLOR 15,14,1
30 SCREEN 2:K=1  (een andere kleur mag ook)
40 R=96
50 FOR P=1 TO 255+30
60 PSET(P,R),K
70 IF P > 30 THEN PRESET(P-30,R)
80 NEXT
90 X$=INPUT$(1)
```

De PRESET-opdracht zonder kleurcode schrijft een punt in dezelfde kleur als de achtergrond. Dat lijkt dus op wissen en omdat de COLOR-opdracht op het grafische scherm niet werkt, mogen we ook best zeggen dat dat zo is. Merk op dat we best "buiten het scherm" mogen tekenen. We zien er alleen niets van.

De MSX kent een aantal opdrachten waarmee getekend kan worden. Probeer:

```
10 CLEAR 1000:CLS
20 COLOR 15,14,1
30 SCREEN 2:K=1
40 P=128:R=96:S=50
50 CIRCLE (P,R),S,K
60 X$=INPUT$(1)
```

Een fraaie, ronde cirkel is het niet. Dat komt door het tv-systeem. Een vierkant met 4 even lange zijden wordt ook niet vierkant. We moeten de cirkel corrigeren, maar daar is al op gerekend. Naast het middelpunt (tussen haakjes hier P en R), de straal (hier S) en de kleur (hier K) kunnen we een beginhoek, een eindhoek en een "kijkrichting" of "aangezichtsverhouding" opgeven:

```
45 B=1:E=1:V=1.4
50 CIRCLE (P,R),S,K,B,E,V
```

B en E kunnen voor een cirkelboog zorgen, V zorgt dat de cirkel rond wordt.

Opmerkingen:
1. Alle variabelen mogen ook als constanten in de CIRCLE-opdracht staan. Hier hebben we er variabelen van gemaakt, omdat dat gemakkelijker is bij het experimenteren.
2. B, E en V mogen weggelaten worden. Deze drie zijn echter wel aan hun plaats gebonden: als alleen E wegblijft, ziet de opdracht er dus zo uit:

   ```
   CIRCLE (P,R),S,B,,V
   ```

Experimenteer met waarden van B en E tussen 0 en 6.28. Op de wiskundige achtergronden daarvan gaan we niet in. Veel aardiger is het om te experimenteren met de "aangezichtsverhouding":

```
10 CLEAR 1000:CLS
20 COLOR 15,14,1
30 SCREEN 2:K=1
40 P=128:R=96:S=80
50 FOR V=0.2 TO 10 STEP 0.4   (.2 en .4 mag ook!)
60 CIRCLE (P,R),S,K,,,V       (let op de komma's)
70 IF K=1 THEN K=6 ELSE K=1
80 NEXT V
90 OPEN "GRP:" FOR OUTPUT AS #1
100 PSET(0,180)
110 PRINT #1,"Klaar!"
120 CLOSE #1
130 X$=INPUT$(1)
```

De moeilijke constructie op regel 90 tot en met 120 is nodig om op het grafische scherm een tekst te kunnen krijgen. De MSX beschouwt het grafische scherm dan als een "apparaat" met de naam "GRP:" (denk aan de dubbele punt) zoals de cassetterecorder CAS: heet. Naar dit apparaaat wordt de tekst geschreven alsof we met een bestand aan het werk zijn. Dit bestand moet dan eerst geopend worden. Een naam is niet nodig, een nummer wel. Dat nummer komt terug in de PRINT -opdracht. De PSET-opdracht geeft de plaats aan (in puntjes-posities!).

Opmerking: doordat de lijnen van de cirkel en de ellipsen dicht bij elkaar komen, kan niet iedere lijn zijn eigen kleur krijgen! De tekst op het scherm krijgt de kleur van de COLOR-opdracht. Voor teksten werkt de COLOR-opdracht wél op het grafische scherm.

Voeg nu toe:

```
65 IF V>=1.4 THEN PAINT (P,R),K,K
```

en zie wat een fraai resultaat! De PAINT schildert een vlak vanuit punt P,R in de kleur K. De rand wordt in de tweede kleur (hier ook K) geschilderd (op SCREEN 2 kan dat niet anders, op SCREEN 3 wel).

Verander nu SCREEN 2 in SCREEN 3 (regel 30) en pas regel 100 aan:

```
100 PSET (0,18)
```

Voer het programma uit en kijk wat er met de tekst "Klaar!" gebeurt.

Zet het scherm weer op 2, verander regel 60 en pas regel 50 aan:

```
60 LINE (P,R)-(P+50,R-V),K
50 FOR V=-30 TO 30 STEP 4
```

Regel 65 (PAINT) mag eruit. Bekijk het resultaat.

LINE trekt een lijn van de eerstgenoemde positie (P,R) naar de tweede positie (P+50,R+V) in de kleur K.

De opdracht DRAW (teken) heeft wat weg van de muziekopdracht PLAY: na DRAW volgt een hele reeks stuurtekens met waarden, waarmee getekend kan worden. Een enkel voorbeeld:

```
10 CLEAR 1000:CLS:SCREEN 2:PSET(128,96)
20 READ A$
30 IF A$="stop" THEN STOP
40 DRAW A$
50 X$=INPUT$(1)
60 GOTO 20

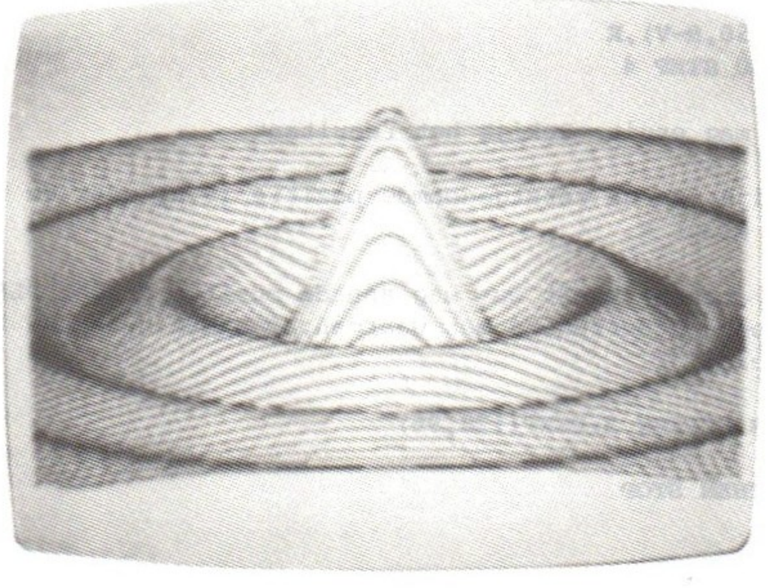
200 DATA c4s8,r10,u20,l10,d20,e10,f10,g10,h10,"m20,20","m+5,+5","m+6,-6",
m-7,7","m-8,-8",stop
```

De betekenis van de stuurcodes is:

c kleur
s grootte van de stapjes (4=1 stip)
r rechts
l links
u naar boven (up)
d naar beneden (down)

Kijk op het scherm wat de andere stuurcodes presteren!

Opmerking: let op de data tussen aanhalingstekens. Deze zijn hier nodig omdat er een komma in het gegeven (de data) zelf staat: "m20,20" is één geheel en moet als één gegeven gelezen worden (betekenis: trek een lijn naar punt 20,20).



116. Onderbreken

We hebben in een eerder hoofdstuk kennis gemaakt met de foutroutine "ON ERROR GOTO...". Deze opdracht diende om bij het optreden van een fout in het programma deze foutsituatie door de computer zelf te laten oplossen.

In de termen van deze paragraaf moeten we zeggen: een fout in het programma zorgt voor een onderbreking (Engels: interrupt) van het programma. Zo'n onderbreking kan worden opgevangen met een ON ERROR-opdracht.

In deze paragraaf maken we kennis met een paar andere onderbrekingen, die op soortgelijke wijze opgevangen kunnen worden met ON ....-opdrachten.

We beginnen met de CTRL-STOP. Dat is een duidelijke vorm van onderbreken. In het programma kunnen we een opdracht opnemen die aangeeft wat er moet gebeuren als CTRL-STOP gebruikt wordt:

```
10 ON STOP GOSUB 100
20 PRINT "*";
30 GOTO 20
100 PRINT
110 PRINT "Ik stop lekker niet!"
120 RETURN
```

Probeer het programma en bekijk het resultaat. Er gebeurt nog niets bijzonders: het programma stopt gewoon met een mededeling "Break in ...". Dat komt omdat op regel 10 weliswaar aangegeven is waar de MSX "even langs" moet als de CTRL-STOP wordt ingedrukt, maar de STOP-toets is nog niet aan deze ON STOP GOSUB gekoppeld. Daarvoor is nog een tweede opdracht nodig.

```
15 STOP ON
```

Deze opdracht zegt: de ON STOP-opdracht is nu ingeschakeld.

Probeer het programma nu opnieuw! We merken al snel dat CTRL-STOP nu niet meer werkt. Dat is jammer, want er is nu nog maar één manier om het programma te stoppen, namelijk met RESET. En dat kost ons wel ons programma!

De ON STOP GOSUB-opdracht kan in het programma ook weer uitgeschakeld worden:

```
10 ON STOP GOSUB 100
20 STOP ON
30 FOR I=1 TO 500
40 PRINT I;
50 NEXT
60 STOP OFF
70 GOTO 30

100 PRINT
110 PRINT "stoppen kan pas na 500"
120 RETURN
```

LET OP: STOP ON schakelt niet de STOP-toets in, maar de ON STOP GOSUB-opdracht. STOP OFF schakelt niet de STOP-toets uit, maar de ON STOP GOSUB-opdracht!

Dit programma kan pas gestopt worden als de MSX de gelegenheid heeft gehad tenminste éénmaal tot 500 te tellen.

Opmerkingen:

1. Anders dan bij de "gewone" RETURN kan in een RETURN na een ON STOP GOSUB-opdracht een regelnummer vermeld worden:

   120 RETURN 30 (na CTRL-STOP begint de MSX opnieuw bij 1 te tellen)

   Denk er echter aan dat een subroutine altijd met een RETURN verlaten moet worden en nooit met een GOTO-opdracht
2. De pauzeer-mogelijkheid van STOP blijft altijd werken, alleen CTRL-STOP kan "afgevangen" worden.
3. De ON STOP GOSUB mag overal in het programma staan. De MSX onthoudt wat hij bij CTRL-STOP moet doen uit de laatste ON STOP GOSUB-opdracht die hij heeft uitgevoerd.

Vergelijkbaar met de ON STOP GOSUB en de daarbij behorende STOP ON- en STOP OFF-opdrachten, zijn de

ON KEY GOSUB met daarbij KEY (...) ON en KEY (...) OFF

en

ON INTERVAL GOSUB met INTERVAL ON en INTERVAL OFF

Met de ON KEY kunnen we reageren op het indrukken van één van de tien functietoetsen F1 tot en met F10. De regels zijn gelijk aan die bij de ON STOP.

```
10 ON KEY GOSUB 200,300,400,500,600
20 FOR I=1 TO 5:KEY(I) ON:NEXT
30 COLOR 1,15,4
40 SCREEN 2:K=1:S=60
50 OPEN "GRP:" FOR OUTPUT AS #1
60 PSET(0,180)
70 COLOR 15
80 PRINT #1,"geel,rood groter kleiner wis"
90 CLOSE #1
100 CIRCLE (128,96),S,K,,,1.4
110 PAINT (128,96),K
120 GOTO 100

200 K=11:RETURN 100
300 K=6:RETURN 100

400 IF S >120 THEN RETURN 100
410 IF K <>1 THEN K=1 ELSE K=13
420 S=S+10
430 RETURN 100

500 IF S < 10 THEN RETURN 100
510 IF K <>1 THEN K=1 ELSE K=15
520 S=S-10
530 RETURN 100

600 CLS:K=1:RETURN 40
```

Op regel 10 is aangegeven naar welke subroutines de MSX moet springen als er op één van de functietoetsen F1 tot en met F5 gedrukt wordt. Op regel 20 worden deze vijf functietoetsen ingeschakeld. Op het grafische scherm kunnen cirkels getekend worden. De straal van de begincirkel is 60, de kleur is zwart. Deze cirkel wordt op regel 110 gekleurd in de kleur K (de kleuren van de cirkel zelf en van de "verf" moeten gelijk zijn)
Op regel 70 tot en met 90 wordt de betekenis van de vijf functietoetsen op het scherm gezet: met F1 maken we de cirkel geel, met F2 rood. Met toets F3 maken we een grotere cirkel, met F4 een kleinere. Met F5 wissen we het scherm.

De MSX tekent steeds dezelfde cirkel in dezelfde kleur, totdat op één van de vijf functietoetsen gedrukt wordt. De computer maakt dan een uitstapje naar de bijbehorende routine.

- F1 : de MSX gaat langs bij 200, verandert de kleur K in geel en keert terug naar de regel waar de cirkel getekend wordt
- F2: idem, maar nu in rood (regel 300)
- F3: de MSX gaat langs bij 400. Daar wordt eerst gekeken of de cirkel niet té groot wordt. Als dat het geval is gaat hij terug naar 100. Als de cirkel nog best wat groter kan, kijkt de MSX eerst naar de kleur van de vorige cirkel: als die niet zwart was wordt de volgende zwart, was die wél zwart dan wordt de volgende paars. De straal wordt 10 stapjes groter. De MSX keert terug naar de cirkel-opdracht.

F4: idem, maar nu met een kleinere straal (regel 500)
F5: de MSX wist het scherm en maakt de beginkleur weer zwart. Daarna keert hij terug naar regel 50, omdat door de CLS-opdracht ook de tekst onderaan op het scherm gewist is.

Opmerkingen:

1. Bij het gebruik van onderbrekingen met functietoetsen kunnen ongewenste effecten optreden, die in het programma opgeheven moeten worden. In dit programma blijkt zo'n probleem bijvoorbeeld als we één van de toetsen F3 of F4 (grotere, resp. kleinere cirkel) ingedrukt houden. De toets is dan ook ingedrukt tijdens de uitvoering van de onderbrekingsroutine. Daardoor krijgt de MSX geen gelegenheid de cirkels te kleuren en ontstaan niet-ingevulde cirkels in of om elkaar.
   Dit effect kan worden opgeheven door aan het begin van een onderbreekroutine de functietoetsen uit te schakelen en ze aan het eind weer "aan" te zetten. Er is ook een "vasthoudmogelijkheid". Zie daarvoor het BASIC-boekje bij de computer.
2. De functietoetsen worden hier anders gebruikt dan bij de opdracht KEY 1,"geel". Met deze opdracht wordt een tekst (of een getal) aan F1 "gehangen", zodat die gemakkelijk in te voeren is bij een INPUT-opdracht. Bij een INPUT-opdracht wacht de MSX tot iets ingevoerd is en gaat dan verder met het programma. Bij gebruik van de ON KEY-opdracht gaat de MSX door met het uitvoeren van het programma en maakt pas een uitstapje als op één van de "vrijgegeven" (ingeschakelde) functietoetsen gedrukt is. Deze twee mogelijkheden kunnen best in één programma gebruikt worden voor dezelfde toets, maat niet tegelijkertijd. De ON KEY gaat vóór en daarom moet de INPUT-opdracht voorafgegaan worden door een KEY (...) OFF en eventueel gevolgd door een KEY (...) ON, bijvoorbeeld:

```
25 CLS:KEY 3,"jantje"+CHR$(13)
35 KEY(3) OFF:INPUT"Typ een naam";A$:KEY(3) ON
85 PSET(0,0):PRINT #1,A$
```

   Met deze constructie kan een naam ingevoerd worden, die op het grafische scherm links bovenaan verschijnt.
   Bij de INPUT-opdracht kan met F3 de naam "jantje" ingevuld worden (door de CHR$(13) is RETURN zelf niet nodig!).

De laatste ON ... GOSUB die we zullen proberen is de ON INTERVAL GOSUB. Met deze opdracht laten we het "langsgaan bij een subroutine" afhangen van de tijd.

```
10 ON INTERVAL=60 GOSUB 100
20 INTERVAL ON
30 COLOR 1,14,6:SCREEN 2:K=1
40 FOR P=0 TO 255 STEP 8:LINE(129,96)-(P,O),K:NEXT
50 FOR R=0 TO 192 STEP 8:LINE(255,96)-(128,R):NEXT
60 FOR P=255 TO 0 STEP -8:LINE(128,96)-(P,192):NEXT
70 FOR R=192 TO 0 STEP -8:LINE(0,96)-(128,R):NEXT
80 GOTO 40

100 K=K+1:IF K=16 THEN K=1
110 RETURN
```

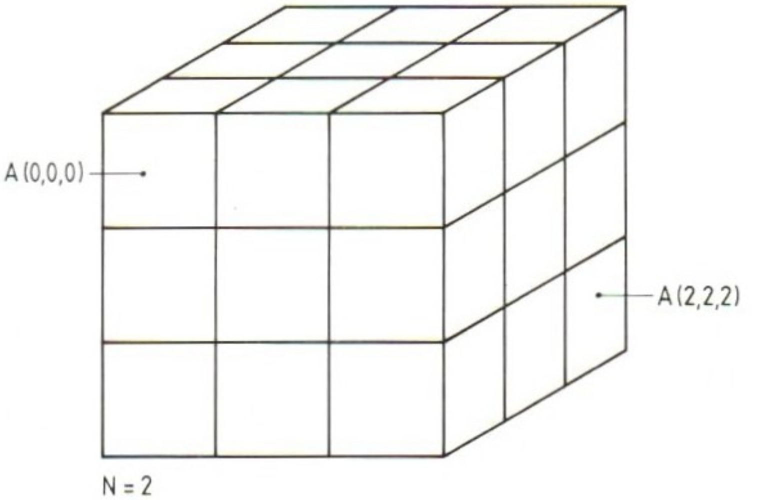
De "eenheid van interval" is 1/60 seconde (ongeveer); bij ON INTERVAL=60 gaat de MSX dus om de seconde even langs bij 100, verandert de kleur en keert weer terug.

117. De laatste probeersels

We besluiten dit hoofdstuk met een reeks experimenten, waarin we BASIC-opdrachten gebruiken, die in dit boek nog niet behandeld zijn. De experimenten laten alleen zien, dat er nog veel meer mogelijk is. De toelichting is kort: meer informatie staat in het BASIC-boekje bij de MSX. Experimenteer zelf verder met de voorbeelden, maar gebruik het BASIC-boekje erbij!

Uit het volgende blijkt dat tabellen meer dimensies mogen hebben.

```
10 CLEAR 1000
20 INPUT "GEEF N";N:IF N >9 THEN 10
30 DIM A$(N,N,N)
40 B=1
50 FOR I=1 TO N
60 FOR P=1 TO N
70 FOR Q=1 TO N
80 A$(I,P,Q)=CHR$(64+B)
90 NEXT Q
100 NEXT P
110 B=B+1
120 NEXT I
130 FOR I=1 TO N
140 FOR P=1 TO N
150 FOR Q=1 TO N
160 PRINT A$(I,P,Q);
170 NEXT Q
180 PRINT
190 NEXT P
200 PRINT
210 NEXT I
```



N = 2

Het is soms erg lastig om overzichten behoorlijk op het scherm te krijgen. Dat ligt niet aan de MSX, maar aan de programmeertaal BASIC. In dit experiment gebruiken we de opdracht PRINT USING. Neem het volgende programma over in de MSX:

```
10 CLS
20 FOR I=1 TO 10
30 X=RND(4)*10
40 PRINT X TAB(15);
50 PRINT USING "##.#";X      (let op de ;)
60 NEXT
```

Vergeet in regel 50 de puntkomma niet. Voer het programma uit en bestudeer het effect van de PRINT USING. Vervang daarna de string in de PRINT USING-opdracht door één van de volgende:

a. ##
b. ##.##
c. ##.###

Kijk vooral naar het afronden. Met de PRINT USING-opdracht kunnen we een getal afgerond op het scherm krijgen. In het geheugen blijft het getal zoals het was (dus niet afgerond).

In de PRINT USING kunnen we de "opmaak" van een regel vastleggen. Met # geven we aan, waar de MSX cijfers moet invullen:

```
10 CLS:PRINT "De tafel van ....."
20 INPUT "Welke tafel";T
30 FOR I=1 TO 10
40 PRINT USING "## x ## = ###"; I;T;I*T
50 NEXT
60 PRINT
70 PRINT "druk op een toets ";
80 X$=INPUT$(1)
90 GOTO 10
```

Bij het vergelijken van twee dingen bepaalt de MSX een waarde: als de uitkomst van een vergelijking "waar" is, is die waarde -1, "niet waar" wordt 0. Met de uitkomst van een vergelijking kunnen we de MSX dus laten rekenen:

```
A=5:B=6
PRINT A=5
PRINT A<5
PRINT A=B
PRINT (A=B) * (A>5)
```

Probeer dan deze eens:

```
10 FOR A=-2 TO 2:PRINT "A="A;
20 IF A THEN PRINT "waar" ELSE PRINT "onwaar"
30 NEXT
```

Erg handig zijn de functies in BASIC. Daarmee kunnen we zelf een functie opgeven. Probeer:

```
10 DEF FNA(X)=3*X
20 INPUT "Getal";Y
30 PRINT FNA(Y)
40 GOTO 20
```

In de definitie op regel 10 staat, hoe de functie A uitgerekend moet worden. Tussen haakjes staat het "Argument", hier X.
Deze X is geen variabele, maar een zogenaamde parameter. Op regel 30 wordt de gedefineerde functie gebruikt in een PRINT-opdracht. De letter tussen haakjes is hier wél een variabele naam. De opdracht zegt: laat de uitkomst van de functie A zien, berekend met als parameter (argument) de waarde van de variabele Y.

Functies zijn vooral goed bruikbaar als een berekening meer dan eens in een programma voorkomt. De functie vervangt dan een subroutine en zorgt voor een overzichtelijker programma:

```
10 DEF FNRAD(G)=(3.1415926536/180/G))
20 INPUT "Op hoeveel graden beginnen";B
30 INPUT "Op hoeveel graden stoppen";E
40 SCREEN 2
50 CIRCLE (180,96),80,1,FNRAD(B),FNRAD(E),1.4
60 CIRCLE (40,96),80,6,FNRAD(-B),-FNRAD(E),1.4
70 X$=INPUT$(1)
80 GOTO 20
```

In dit programma wordt een cirkelboog getekend. De begin- en eindhoek moet in de CIRCLE-opdracht in radialen opgegeven worden. Dat vraagt voor in graden denkende mensen een omrekening, die in de functie op regel 10 geprogrammeerd is.
Op regel 50 wordt deze functie tweemaal aangeroepen: éénmaal met de variabele B als argument (beginhoek), éénmaal met de variabele E als argument (eindhoek). Voor dit soort omrekenprobleempjes zijn functies ideaal!

De hier gebruikte functie stelt een getal voor, waarmee ook verder gerekend kan worden (regel 60 bewijst dat!). Een functie kan ook een string opleveren, maar dan moet de naam eindigen met een dollar-teken:

```
10 CLEAR 1000
20 DEF FNNM$(A$)="Hallo, "+A$+"!"
30 INPUT "Hoe heet je";A$
40 PRINT FNNM$(A$)
```

Met de functie FN ("argument") wordt in dit programma steeds de tekst "Hallo, argument!" opgeroepen. Het argument mag als constante (tekst, getal) tussen de haakjes staan, maar ook als variabele, berekening of zelfs als functie. Probeer maar:

```
40 PRINT FNNM$(FNNM$(A$))
```

Kijk naar het resultaat.

Hiermee besluiten we het **BASIC-probeerboek.** Er is nog veel meer te proberen en te leren, maar dat moet lukken met het MSX-BASIC-boekje als basis!

Bijlage A: Oplossingen

Hoofdstuk 1

Opdracht 1

a) k	d) K	g) K	j) K
b) 7	e) &	h) 7	k) &
c) ;	f) :	i) ;	l) :

Opdracht 2

a) 8	e)
b) 8	f)
c) *	g)
d) *	h)

Opdracht 3

Scherm schoonmaken: CLR (SHIFT HOME)
Loop met de cursortoetsen naar de plaats waar het hok moet komen.
Typ de bovenkant met GRAPH en de streep.
Ga dan steeds met de cursortoetsen één regel naar beneden en één posite naar links. Typ met GRAPH een verticale streep. Maak zo de rechter- en linker zijkant van het hok.
Tenslotte is de bodem aan de beurt.

Hoofdstuk 2

Opdracht 1

```
Wij: print 73300/(26*2.54*3.142)
MSX: 353.25696171638
```

Opdracht 2

```
Wij: print (3300/8.9-3300/12.2)*1.64
MSX: 164.48333026341
```

Opdracht 3

```
Wij: print 45/11
MSX: 4.0909090909091
Wij: print 11/45
MSX: .24444444444444
```

Let op! Als de antwoorden niet helemaal hetzelfde zijn maar wel heel dicht in de buurt komen kunnen de verschillen ontstaan zijn door de afronding.

Hoofdstuk 4

Opdracht 1

BA$, BAL$ en BAR$ beginnen met BA en zijn voor de MSX dus gelijk.
POP is fout, omdat het dollarteken $ vergeten is.
4Q$ is fout, omdat deze naam met een cijfer begint.

Opmerking: de naam POP kunnen we later wel voor een ander soort vakje gebruiken.

Opdracht 2

```
Wij: 6H$="WIM"
Wij: PRINT 6H$
MSX: 6
     Ok
Wij: RUN
MSX: Ok
```

Als we een regel laten beginnen met een cijfer is dat automatisch een regelnummer. De MSX beschouwt de 6 die we als eerste getypt hebben als een regelnummer, ook al dachten wij dat die 6 bij de naam hoorde. We hebben dus een regel van een programma ingetypt en daarmee gaat de MSX akkoord.
Hij reageert daar niet op maar wacht tot een RUN-opdracht gegeven wordt om deze programmaregel uit te voeren. Als we daarna direct een PRINT-opdracht geven voert de MSX deze PRINT-opdracht wél uit. Hij drukt de 6 af en de string H$. De 6 komt op het scherm. In H$ staat niets (want regel 6 is nog niet uitgevoerd) dus daarvoor komt ook niets op het scherm te staan. Als we vervolgens de RUN-opdracht geven voert de MSX de programmaregel met nummer 6 uit. Omdat deze regel alleen een string in een vakje zet merken we daar verder niets van. De MSX reageert alleen met Ok omdat hij zijn werk gedaan heeft. Als we vervolgens PRINT 6H$ geven reageert de MSX met: 6 WIM omdat regel 6 inmiddels wél uitgevoerd is.

Opdracht 3

```
10 A$=" GRAS "
20 B$=" GROEN "
30 C$="IS"
40 PRINT A$ C$ B$ "*";
50 D$=A$
60 A$=B$
70 B$=D$
80 GOTO 40
```

Bekijk dit programma. Aan de voorwaarden in het experiment is voldaan: GRAS en GROEN staan maar éénmaal in het programma en er komt maar één PRINT-opdracht in voor. De regels 50, 60 en 70 dienen om GRAS en GROEN van plaats te laten wisselen. Daarvoor hebben we wel een extra vakje nodig (D$).

Opdracht 4

```
10 A$="HANS
20 B$="GRIET"
30 PRINT A$ " EN " B$ " IS ";
40 PRINT B$ "JE EN " A$ "JE"
50 GOTO 30
```

Op alle plaatsen waar we het programma verbeterd hebben is de verbetering onderstreept. Regel 10, waarin een aanhalingsteken ontbreekt is niet fout.

Hoofdstuk 6

Opdracht 1

```
Hetzelfde zijn: BA
                BAL
                BART
                BAS

Fout zijn:      2A      (begint met een cijfer)
                JA$     (is stringvariabele)
                ANTON   (er zit TO in de naam)
                A$      (is een stringvariabele)
```

Opdracht 3

```
10 CLS
20 INPUT "LENGTE";A
30 INPUT "BREEDTE";B
40 PRINT "OPPERVLAKTE IS " A*B " M2"
50 PRINT
60 GOTO 20
```

Dit is niet de enige goede oplossing. Als het programma werkt is de oplossing goed, ook al ziet hij er anders uit dan dit voorbeeld.
In experiment 3 stopt de MSX als de uitkomst van het sommetje A=A*B te groot wordt. We krijgen dan "overflow". Dat betekent dat de uitkomst van de som te groot is om in vakje A te zetten.

Opdracht 5

In opdracht 5 kan het heel lang duren voordat de MSX stopt. Hij doet dat als hij "overflow" krijgt, net als in experiment 3, maar het getal B groeit veel langzamer aan. Het zit er dus dik in dat we het programma al lang voor die tijd gestopt hebben met de STOP-toets. Regel 30 in dit programma dient er alleen voor de MSX na elke telopdracht te laten stoppen en pas weer door te laten gaan als wij iets getypt hebben.
Wat dat is, is niet belangrijk: met wat wij typen in de stringvariabele B$ doet het programma niets.
Regel 30 is dus alleen een "stop-punt". Alleen de regeltoets aanslaan is voldoende.

Hoofdstuk 7

Vraag 1 : Het juiste antwoord is 20. Om 20 sterren te krijgen moet P lopen van 1 t/m 20.

Vraag 2 : Het goede antwoord is 14. Op regel 20 worden twee sterren achter elkaar afgedrukt. Om 10 sterren te krijgen moet dit vijf keer gebeuren. Als Q begint bij 10, wordt regel 20 vijf keer uitgevoerd, als Q doorloopt t/m 14. Q neemt achtereenvolgens de waarden 10, 11, 12, 13 en 14 aan. Dat is precies vijf keer.

Vraag 3 : Het volgende schema laat zien wat er met R en de sterren gebeurt:

RONDE	R	SCHERM
1	20	*
2	18	*
3	16	*
4	14	*
5	12	*
6	10	*
7	8	mag niet meer.

Het juiste antwoord is dus 6.

Opdracht 3

In opdracht 3 wordt op regel 70 gesprongen naar regel 30. Dat is midden in een herhaling met FOR-NEXT. Bij de uitvoering komt de MSX de FOR-regel niet tegen, maar wel de NEXT.
Dat levert een foutbericht "Next without for" op.

Opdracht 4

```
10 CLS
20 INPUT "AANTAL GETALLEN";A
30 P=1
40 FOR I=1 TO A
50 INPUT "GETAL";G
60 P=P*G
70 NEXT I
80 PRINT "HET PRODUKT VAN DE" A "GETALLEN IS" P
```

Een andere oplossing kan ook best goed zijn. Controleer bijvoorbeeld met 30*40*20=24000

Opdracht 5

```
10 CLS
20 PRINT "GETAL" TAB(10) "HELFT" TAB(20) "DERDE" TAB(30) "VIERDE"
30 FOR G=100 TO 1000 STEP 100
40 PRINT G;
50 PRINT TAB(8) G/2;
60 PRINT TAB(16) G/3;
70 PRINT TAB(32) G/4
80 NEXT G
```

In plaats van de PRINT-opdrachten op regel 40, 50, 60 en 70 kunnen we ook één PRINT opdracht op regel 40 zetten:

```
40 PRINT G TAB(10) G/2 TAB(20) G/3 TAB(30) G/4
```

Dan kunnen we kijken hoe het programma werkt en of de kolommen netjes op het scherm komen. Daarna kunnen we de TAB posities aanpassen aan onze wensen. De lengte van de getallen maakt dat overigens niet eenvoudig!

Opdracht 6

Een oplossing voor opdracht 6 kan er zo uitzien:

```
10  CLS
20  FOR R=1 TO 4
30  PRINT
40  NEXT R
50  FOR R=1 TO 3
60  FOR P=1 TO 40
70  PRINT "*";
80  NEXT P
90  NEXT R
100 GOTO 20
```

Opmerking: In deze oplossing hebben we de TAB-functie niet gebruikt. Dat is ook niet nodig, maar het mag natuurlijk wel.

Hoofdstuk 8

Hier komen de antwoorden op de vraag. Achter elk antwoord staat op welke regels het programma geweest is t/m de print die de tekst op het scherm zet.

a.	16	LEKKER	(40-50-60-90-120-130)
b.	18	LEKKER	(40-50-60-90-120-130)
c.	-4	KOUD	(40-50-60-70)
d.	0	KOUD	(40-50-60-70)
e.	34	WARM	(40-50-60-90-120-150)
f.	8	KOUD	(40-50-60-70)
g.	14	FRIS	(40-50-60-90-100)
h.	60	EINDE	(40-170)
i.	22	LEKKER	(40-50-60-90-120-130)
j.	-50	EINDE	(40-50-170)
k.	10	FRIS	(40-50-60-90-100)

Opdracht 1

```
10 CLS
20 INPUT "AANTAL INCHES";I
30 IF I=0 THEN 60
40 PRINT I "INCH IS" I*2.54 "CM"
50 GOTO 20
60 PRINT "GESTOPT"
```

Het programma stopt als we op regel 20 voor I nul typen.

Opdracht 2

```
10 CLS
20 INPUT "LENGTE";L
30 IF L=0 THEN 120
40 IF L<0 THEN PRINT "NEGATIEF KAN NIET": GOTO 20
50 INPUT "BREEDTE";B
60 IF B=0 THEN PRINT "BREEDTE 0 KAN NIET": GOTO 50
70 IF B<0 THEN PRINT "NEGATIEF KAN NIET": GOTO 50
80 PRINT "OMTREK IS 2X" L "+2X" B "=" 2*L+2*B
90 PRINT "OPPERVLAKTE IS" L "X" B "=" L*B
100 PRINT
110 GOTO 20
120 PRINT "STOP"
```

Dit programma stopt als we voor de lengte L nul typen.
Als we bij de breedte B nul typen, vindt de MSX dat niet goed.

Opdracht 3

```
10 CLS
20 S=0:P=1:T=0                          (T en S beginnen op 0,  P op 1)
30 INPUT "GETAL";G
40 IF G=0 THEN 90                       (Getal 0 betekent: stop)
50 T=T+1                                (Tel het getal)
60 P=P*G                                (Vermenigvuldig P met het getal)
70 S=S+G                                (Tel het getal op bij S)
80 GOTO 30
90 PRINT "AANTAL GETALLEN:"T
100 PRINT "PRODUCT IS" P
110 PRINT "SOM IS" S
120 PRINT "GEMIDDELDE IS";               (Blijf op dezelfde regel)
130 IF T=0 THEN PRINT 0 ELSE PRINT S/T
140 PRINT
150 PRINT "KLAAR"
```

Op regel 130 is het delen-door-nul-probleem opgelost. Als het aantal getallen (T) nul is, drukken we meteen nul af; is T niet nul, dan delen we de som door het aantal getallen.

Hoofdstuk 9

Opdracht 1

```
70 A=INT(RND(4)*100*ABS(B))+3
```

Opdracht 2

```
200 D=B/2.5
210 D=INT(D)
220 E=C/0.3
230 E=INT(E)
240 F=B-C
250 F=ABS(F)
260 G=D+E
270 H=G/F
280 I=4*H
290 A=INT(I)
```

Opmerking: We hebben in deze oplossing voor de duidelijkheid meer letters gebruikt dan nodig is!

Opdracht 3

De functie INT neemt het eerste gehele getal dat gelijk is aan of kleiner is dan het argument. Deze functie hebben we al enkele malen gebruikt. De functie FIX geeft ons de cijfers voor de punt. Wat achter de punt staat valt af. De functie CINT geeft ons een afgerond geheel getal. In de meeste gevallen zal de functie CINT dus beter bruikbaar zijn dan de functie INT.

Hoofdstuk 10

Opdracht 1

GOSUB-RETURN
ON ERROR-RESUME NEXT
FOR-NEXT
DATA-READ en RESTORE
IF-THEN en ELSE

Opdracht 2

In deze oplossing mogen negatieve getallen getypt worden. Daarmee kunnen we corrigeren. Als we bijvoorbeeld 100 liter hebben getypt in plaats van 10 kunnen we op de volgende regel -90 liter typen. De fout is dan hersteld.
In deze oplossing hebben we bovendien een waarschijnlijkheidscontrole opgenomen. We mogen verwachten dat de uitkomst van de berekening voor een normale personenauto zal liggen tussen de 5 en de 20. Een auto die meer benzine gebruikt, of één die nog zuiniger is dan 1 op 20, is erg onwaarschijnlijk.
Als we dit programma maken voor een motor of bromfiets zullen we de grenzen natuurlijk anders moeten leggen.

```
10 CLS
20 T=0
30 READ B
40 READ E
50 IF B>=E THEN PRINT "KM-STANDEN FOUT; HERHAAL":GOTO 30
60 K=E-B
70 PRINT "VERREDEN"K"KM"
80 PRINT
90 PRINT "GETANKT"
100 READ L:PRINT L;
110 IF L<0 THEN PRINT "KORREKTIE" ELSE PRINT
120 IF L=0 THEN 150
130 T=T+L
140 GOTO 100
150 PRINT "AANTAL KM/LITER:" K/T;
160 IF K/T<5 OR K/T>20 THEN PRINT "UITKOMST ONWAARSCHIJNLIJK" ELSE PRINT
170 GOTO 20

500 DATA 55000,56000,35,35,30,0
510 DATA 55000,54000
520 DATA 55000,56000,35,40,35,38,40,30,0
530 DATA 55000,56000,350,-350,35,35,30,0
```

Opmerking: Op regel 160 is een gecombineerde vergelijking gebruikt:

IF K/T < 5 OR K/T > 20 THEN .... enz.

De betekenis is: als ... of ... dan ..., waarbij de MSX verder gaat achter THEN als aan ten minste één van beide voorwaarden (hier: K/T kleiner dan 5 of K/T groter dan 20) is voldaan.
Merk op, dat achter OR (of) het woord IF niet herhaald wordt.

Hoofdstuk 11

Opdracht 2

Voorbeeld: A$="ABCDEF" en B$="123456"

Opdracht	Resultaat in A$
A$=MID$(B$,4,2)	45
MID$(A$,4,2)=B$	ABC12F
MID$(A$,4,2)=MID$(B$,2,2)	ABC23F
A$=LEFT$(B$,2)	12
LEFT$(A$,2)=B$	Syntax error
LEFT$(A$,2)=LEFT$(B$,2)	Syntax error
A$=RIGHT$(B$,2)	56
RIGHT$(A$,2)=B$	Syntax error
RIGHT$(A$,2)=RIGHT$(B$,2)	Syntax error
MID$(A$,3,2)=RIGHT$(B$,2)	AB56EF
MID$(A$,3,2)=LEFT$(B$,2)	AB12EF

Kort gezegd is dus de conclusie:
De LEFT$ en de RIGHT$ kunnen we gebruiken om iets links, resp. rechts uit een string te halen, niet om iets op die plaatsen in een string te zetten. Dat kan alleen met de MID$.

Opdracht 3

In opdracht 3 maken we gebruik van de ASCII-codes van de letters.
Op regel 20, 30 en 40 zoeken we de hele string A$ af. Elke keer als we een letter tegenkomen met een ASCII-code tussen 96 en 123 (dat zijn de codes 97 t/m 122, dus de kleine letters a t/m z) halen we van die code 32 af.
De kleine letter op die positie wordt dan een hoofdletter.

```
10 INPUT A$
20 FOR P=1 TO LEN(A$)
30 IF ASC(MID$(A$,P,1))  96 AND ASC(MID$(A$,P,1))  123 THEN MID$(A$,P,1)=
CHR$(ASC(MID$(A$,P,1))-32)
40 NEXT P
50 PRINT A$
```

De MSX gaat dan zo aan het werk:

```
MID$(A$,P,1)                 is letter n
ASC(MID$(A$,P,1))            is code van n is 110
ASC(MID$(A$,P,1))-32         is 110-32 is 78
CHR(ASC(MID$(A$,P,1))-32)    is letter bij code 78 is dus N
```

dus: MID$(A$,P,1) wordt N in plaats van n

Opmerking: op regel 30 staat een gecombineerde vergelijking met AND in de vorm: als ... én ... dan enz.

De MSX gaat alleen verder achter THEN, als het antwoord op beide vergelijkingen ja is. Als het antwoord op een van beide of beide vergelijkingen nee is, is aan de gestelde voorwaarde niet voldaan en gaan we verder op de volgende regel (of achter ELSE, als dat er staat).

Merk op, dat het woord IF niet herhaald wordt.

Hoofdstuk 12

Opdracht 1

	tabel	vakje
a.	A	5
b.	B	12
c.	C	24
d.	P$	3
e.	A	5
f.	B	7
g.	C	12
h.	P$	3
i.	A	5
j.	B	9
k.	C	16
l.	P$	8

Opmerking: we kunnen tabellen maken met getallen en tabellen met teksten. Bij tabellen met teksten (strings) moeten we een naam gebruiken met een dollarteken aan het eind.

Opdracht 2

Antwoord b. is fout omdat we niet een tabel-element op het scherm zetten maar het getal in het tabel-element vermenigvuldigen met 10.Antwoord d. is fout omdat we het nummer van een element op het scherm zetten en niet de inhoud van het element zelf. De antwoorden a. en c. zijn goed.

Opdracht 3

We kunnen dat op verschillende manieren oplossen. Dit is de eerste:

```
100 FOR I=1 TO 60 STEP 3
110 PRINT A(I);A(I+1);A(I+2)
120 NEXT
```

Dit is de eenvoudigste manier die goed werkt als we niet te veel getallen naast elkaar hoeven te zetten. We gaan in stapjes van drie door de tabel heen op regel 100.
Op regel 110 zetten we dan elke keer drie getallen op het scherm. Het eerste, het tweede en het derde uit het groepje. Deze oplossing werkt niet zo goed als we veel getallen naast elkaar moeten zetten. We kunnen dan beter één van de volgende oplossingen gebruiken:

```
100 T=0
110 FOR I=1 TO 60
120 PRINT A(I);
130 T=T+1:IF T=3 THEN PRINT:T=0
140 NEXT
```

In deze oplossing gebruiken we een variabele T om te tellen hoeveel getallen we naast elkaar gezet hebben. T begint op 0. Na de eerste drie getallen staat T op 3.
Op regel 130 schuiven we dan een regel op en laten T weer op 0 beginnen. Als er meer getallen naast elkaar komen hoeven we eenvoudig de 3 in 130 te veranderen in een ander getal.
Ook dit is een oplossing:

```
100 FOR B=1 TO 58 STEP 3
110 FOR I=B TO B+2
120 PRINT A(I);
130 NEXT I
140 PRINT
150 NEXT B
```

Opdracht 4

De eenvoudigste oplossing voor dit probleem ziet er zo uit:

```
100 FOR I=1 TO 20
110 PRINT A(I);A(I+20);A(I+40)
120 NEXT
```

Dit is een andere oplossing:

```
100 FOR I=1 TO 20
110 FOR B=0 TO 40 STEP 20
120 PRINT A(I+B);
130 NEXT B
140 PRINT
150 NEXT I
```

Opdracht 5

We moeten hier een oplossing zien te vinden waarmee we uitmaken in welk element van de tabel T we moeten tellen als we een bepaalde letter tegenkomen. We kunnen natuurlijk een oplossing maken met veel vergelijkingen erin. Bij elke letter gaan we dan uitzoeken of het een A, een B enz. is. Daarvoor zijn 51 vergelijkingen nodig.

Het idee is bijna hetzelfde als in het dobbelsteen-programma als we gebruikmaken van de ASCII-code van de letters. We gaan uit van de volgende regel:

tellen in T(I) kan als I gelijk wordt aan:

ASC(letter)-64 bij hoofdletters en
ASC(letter)-96 bij kleine letters.

Onze oplossing wordt dan:

```
10 CLEAR 1000
20 DIM T(26)
30 FOR I=1 TO 26:T(I)=0:NEXT
40 CLS
50 INPUT "Tekst of #";T$
60 IF RIGHT$(T$,1)="#" THEN 120
70 FOR P=1 TO LEN(T$)
80 L=ASC(MID$(T$,P,1))
90 IF L>64 AND L<91  THEN T(L-64)=T(L-64)+1
100 IF L>96 AND L<123 THEN T(L-96)=T(L-96)+1
110 NEXT:GOTO 50
120 PRINT:PRINT "Resultaat"
130 FOR I=1 TO 26 STEP 2
140 PRINT CHR$(I+64)"="T(I) TAB(15)CHR$(I+65)"="T(I+1)
150 NEXT
160 INPUT "Doortellen(J/N)";T$
170 IF T$="J" OR T$="j" THEN 40 ELSE 30
```

Opmerking: als we van de opdracht INPUT op regel 50 een opdracht LINE INPUT maken kunnen we ook teksten met komma's typen. Het vraagteken verschijnt dan niet.
Op de regels 90 en 100 staat een combinatie van vergelijkingen met AND. De betekenis is: als ... en ... dan
Alleen als aan beide voorwaarden voldaan is gaat de MSX achter THEN verder.

Register

Fotoverantwoording

Brandsteder Electronics BV – Badhoevedorp
Philips Telecommunicatie en
Informatie-systemen BV – 's-Gravenhage
Foto's: H.L. van Harrevelt – Leersum

92067

Educaboek